Manual de lenguaje 4D

- Introducción
- Presentación del lenguaje
- Depurador
- Acceso a los objetos de desarrollo
- Área web
- Arrastrar y soltar
- Arrays
- Backup
- BLOB
- Booleanos
- Búsquedas
- Cadenas de caracteres
- Cliente HTTP
- Colecciones
- Compilador
- Comunicaciones
- Conjuntos
- Control de entrada
- Corrector ortográfico
- Documentos del sistema
- Entorno 4D
- Entorno del sistema
- Entrada de datos
- Estructura
- Eventos de formulario
- Fechas y horas
- Formularios
- Formularios de usuario
- Fórmulas
- Funciones estadísticas
- Funciones matemáticas
- Gestión de la caché
- Gráficos
- Grupos y usuarios
- Herramientas
- Imágenes
- Importación y exportación
- Impresión
- Informes rápidos
- Interfaz de usuario
- Interrupciones
- JSON
- LDAP
- Lenguaje
- List Box
- Listas jerárquicas
- Mensajes
- Menús
- Métodos base
- Objetos (Formularios)
- Objetos (Lenguaje)
- Operadores
- ORDA - DataClass
- ORDA - DataClassAttribute
- ORDA - DataStore
- ORDA - Entity
- ORDA - EntitySelection
- PHP
- Portapapeles
- Procesos
- Procesos (Comunicación)
- Procesos (Interfaz de usuario)
- Protocolo seguro
- Recursos
- Registros
- Registros (bloqueo)
- Relaciones
- Selecciones
- Selecciones temporales

- Servicios Web (Cliente)
- Servicios Web (Servidor)
- Servidor Web
- SQL
- Subregistros
- SVG
- Tabla
- Texto multiestilo
- Transacciones
- Triggers
- Variables
- Ventanas
- XML
- XML DOM
- XML SAX
- Lista de temas de constantes
- Códigos de error
- Códigos ASCII
- Novedades
- Comandos obsoletos
- Lista alfabética de los comandos

Introducción

Derechos de autor y menciones legales
Prefacio
Introducción
Construir una aplicación 4D

Derechos de autor y menciones legales

Prefacio

4D tiene su propio lenguaje de programación. Este lenguaje integrado, que comprende más de 1000 comandos, hace de 4D una herramienta muy poderosa para el desarrollo de aplicaciones de bases de datos en ordenadores de escritorio. Puede utilizar el lenguaje 4D para realizar múltiples tipos de tareas, desde la realización de cálculos simples hasta la creación de interfaces de usuario complejas y personalizadas. Por ejemplo, puede:

- Acceder por programación a todos los editores de gestión de registros (order by, query, etc),
- Crear e imprimir informes y etiquetas complejas con los datos de la base,
- Comunicarse con otros sistemas de información,
- Administrar documentos,
- Importar y exportar datos entre bases 4D y otras aplicaciones,
- Incorporar los procedimientos escritos en otros lenguajes en el lenguaje de programación de 4D.

La flexibilidad y el poder del lenguaje de programación de 4D lo convierten en la herramienta perfecta para todos los niveles de usuarios y desarrolladores para alcanzar un amplio rango de tareas de gestión de información. Los usuarios nuevos pueden rápidamente efectuar cálculos. Los usuarios experimentados pueden personalizar sus bases de datos sin tener experiencia en programación. Los desarrolladores experimentados pueden utilizar esta poderosa herramienta de programación para añadir a sus bases funcionalidades sofisticadas, incluyendo transferencia de archivos y comunicaciones. Los desarrolladores con experiencia en programación en otros lenguajes pueden añadir sus propios comandos al lenguaje de 4D.

El lenguaje de programación de 4D se enriquece cuando alguno de los módulos de 4D se añade a la aplicación. Cada módulo incluye comandos de lenguaje que son específicos para las funcionalidades que ofrece.

Sobre los manuales

Los manuales descritos aquí ofrecen una guía sobre las características de 4D y 4D Server. La única excepción es el manual de 4D Server, el cual describe únicamente las características de 4D Server.

- El Manual del Lenguaje es una guía para utilizar el lenguaje de 4D. Utilice este manual para aprender cómo personalizar su base incorporando los comandos y funciones de 4D.
- El Manual de Diseño ofrece descripciones detalladas de las operaciones que puede realizar en el entorno Diseño para crear formularios para manejar los datos.
- El Manual de Iniciación rápida lo lleva a través de lecciones de ejemplo en las cuales usted crea y utiliza una base de datos 4D. Estos ejemplos ofrecen experiencia práctica y ayuda para familiarizarse con los conceptos y características de 4D y 4D Server.
- El Manual 4D Server, el cuál se incluye sólo en el paquete 4D Server, es una guía para manejar bases de datos multiusuario con 4D Server.

Sobre este manual

Este manual describe el lenguaje de 4D. Se asume que usted está familiarizado con términos como tabla, campo, y formulario. Antes de leer este manual, usted debe:

- Utilizar el Manual de Iniciación rápida para trabajar a través de la base de datos de ejemplo.
- Comenzar a crear sus propias bases de datos, consultando el Manual de Diseño cuando sea necesario.
- Aumente sus conocimientos estudiando las numerosas bases de demostración y de ejemplo disponibles en el sitio Web de 4D (http://www.4dhispano.com).

Convenciones de escritura

En este manual, se emplean diferentes convenciones de escritura:

- Siguiendo el ejemplo del editor de métodos de 4D, los comandos se escriben en mayúsculas utilizando caracteres especiales: **CLOSE DOCUMENT**. Las funciones (comandos que devuelven un valor) comienzan con mayúscula inicial y continúan en minúscula: **Change string**.
- En la sintaxis de los comandos, los caracteres { } (llaves) indican los parámetros opcionales. Por ejemplo, **SET DEFAULT CENTURY (siglo{; pivotAño})** significa que el parámetro *pivotAño* puede omitirse cuando se llama el comando.
- En la sintaxis de los comandos, el carácter | indica una alternativa. Por ejemplo, **Table (tablaNum | unPtr)** indica que la función acepta un número de tabla o un puntero como parámetro.
- En algunos ejemplos en esta documentación, una línea de código puede prolongarse en las líneas siguientes por falta de espacio. Sin embargo, debe digitar estos ejemplos como una sola línea de código sin utilizar retornos de carros.

¿A dónde ir desde aquí?

Si está leyendo esté manual por primera vez, lea la sección **Introducción**.

Introducción

Esta sección le presenta el lenguaje de programación de 4D. Se tratan los siguientes temas:

- Qué es el lenguaje y qué puede hacer por usted,
- Cómo utilizar los métodos,
- Cómo desarrollar una aplicación con 4D.

Estos temas se cubren en términos generales; se cubren en más detalle en otras secciones.

¿Qué es un lenguaje?

El lenguaje de 4D no es muy diferente del lenguaje hablado que usamos a diario. Es una forma de comunicación utilizada para expresar ideas, informar, y dar instrucciones. Al igual que un lenguaje hablado, 4D tiene su propio vocabulario, gramática, y sintaxis; que usted utiliza para indicarle a 4D cómo manejar su base y sus datos.

No necesita conocer totalmente el lenguaje para trabajar eficazmente con 4D. Para hablar, usted no necesita conocer por completo el castellano; de hecho, puede tener un vocabulario reducido y sin embargo ser bastante elocuente. Lo mismo sucede con el lenguaje 4D, sólo necesita conocer una pequeña parte del lenguaje para ser productivo, y puede aprender el resto a medida en que sus necesidades aumenten.

¿Por qué utilizar un lenguaje?

A primera vista podría parecer que no hay mucha necesidad de un lenguaje de programación en 4D. En el entorno Diseño, 4D ofrece herramientas flexibles, las cuales no necesitan programación para realizar una gran variedad de tareas de gestión de datos. Tareas fundamentales tales como la entrada de datos, búsquedas, ordenación, y creación de informes son efectuadas fácilmente. De hecho, hay muchas funcionalidades extras disponibles, tales como validación, filtros de validación de datos, representación gráfica y generación de etiquetas.

Entonces ¿Por qué necesitamos un lenguaje 4D? Estos son algunos de sus usos:

- Automatización de tareas repetitivas: estas tareas incluyen modificación de datos, generación de informes complejos y la realización de series largas de operaciones sin la intervención de un usuario.
- Control de la interfaz del usuario: puede manejar ventanas y menús, y controlar los formularios y objetos de la interfaz.
- Gestión de datos de manera sofisticada: estas tareas incluyen proceso de transacciones, validación de datos complejos, gestión multiusuarios, utilización de conjuntos y operaciones de selección.
- Control del ordenador: puede controlar las comunicaciones por el puerto serial, gestión de documentos y de errores.
- Crear aplicaciones: puede crear bases de datos personalizadas, fáciles de utilizar, en entorno Aplicación.
- Añadir funcionalidad al servidor web 4D integrado: crear páginas HTML dinámicas además de las traducidas automáticamente de los formularios por 4D.

El lenguaje le permite tomar el control total del diseño y operación de su base de datos.4D proporciona herramientas "genéricas" poderosas, el lenguaje le ofrece la posibilidad de personalizar su base de datos a cualquier grado que necesite.

Tome el control de sus datos

El lenguaje de 4D le permite tomar el control total de sus datos de una manera poderosa y elegante. El lenguaje es fácil para un principiante y lo suficientemente sofisticado para un desarrollador de aplicaciones experimentado. Le ofrece pasar sin complicaciones de las funciones integradas a una base totalmente personalizada.

Los comandos de lenguaje de 4D le ofrecen la posibilidad de acceder a los editores estandar de gestión de registros. Por ejemplo, cuando utiliza el comando **QUERY**, accede al editor de búsquedas (accesible en el modo Diseño utilizando el comando Query en el menú Registros). Puede indicarle al comando **QUERY** buscar datos descritos explícitamente. Por ejemplo, **QUERY** *([Personas]; [Personas]Apellido="Suárez")* encontrará todas las personas de apellido Suárez en su base de datos.

El lenguaje de 4D es muy poderoso, con frecuencia un comando reemplaza cientos o incluso miles de líneas de código escritas en lenguajes de programación tradicionales. Sorprendentemente, este poder viene con simplicidad, los comandos tienen nombres en inglés sencillo. Por ejemplo, para realizar una búsqueda, se utiliza el comando **QUERY**; para añadir un nuevo registro, se utiliza el comando **ADD RECORD**.

El lenguaje está diseñado para que fácilmente pueda llevar a cabo casi cualquier tarea. Añadir un registro, organizar registro, buscar datos, y operaciones similares se definen con comandos simples y directos. El lenguaje también puede controlar los puertos seriales, leer documentos en el disco, controlar sistemas de transacciones complejas, y mucho más.

El lenguaje 4D puede llevar a cabo incluso las tareas más sofisticadas con relativa simplicidad. Realizar estas tareas sin utilizar el lenguaje sería inimaginable para muchos.

Incluso con los poderosos comandos del lenguaje, algunas tareas pueden ser complejas y difíciles. Una herramienta en sí misma no hace que una tarea sea posible; la tarea puede ser exigente y la herramienta sólo facilita el proceso. Por ejemplo, un procesador de palabras hace que escribir un libro sea más rápido y fácil, pero no escribirá el libro por usted. El lenguaje 4D le permite gestionar más fácilmente sus datos y llevar a cabo tareas complejas con confianza.

¿Es un lenguaje de programación “Tradicional”?

Si está familiarizado con lenguajes de programación tradicionales, esta sección podría interesarle. Si no puede omitirla.

El lenguaje de 4D no es un lenguaje de programación tradicional. Es uno de los lenguajes más innovadores y flexibles disponibles para un computador. Está diseñado para adaptarse a su forma de trabajo y no al contrario.

Para utilizar los lenguajes tradicionales, debe realizar una planeación extensiva. De hecho, la planeación es uno de los pasos principales en un desarrollo. 4D le permite comenzar a utilizar el lenguaje en cualquier momento y en cualquier parte de su base de datos. Puede comenzar por añadir un método a un formulario, más adelante añadir unos pocos métodos más. A medida que su base de datos se vuelve más sofisticada, puede añadir un método de proyecto controlado por un menú. Puede utilizar tanto lenguaje como quiera. No es "todo o nada," como es el caso de muchas otras bases de datos.

Los lenguajes tradicionales lo obligan a definir y predeclarar objetos en forma sintáctica rígida. En 4D, usted simplemente crea un objeto, como un botón y lo utiliza. 4D administra automáticamente el objeto por usted. Por ejemplo, para utilizar un botón, basta con dibujarlo en un formulario y colocarle un nombre. Cuando el usuario hace clic en el botón, el lenguaje notifica automáticamente a sus métodos.

Los lenguajes tradicionales con frecuencia son rígidos e inflexibles, y necesitan que los comandos se introduzcan en un estilo muy formal y restrictivo. El lenguaje 4D rompe con la tradición y los beneficios son para usted.

Los métodos son la puerta hacia el lenguaje

Un método es una serie de instrucciones que provocan la realización de una tarea por parte de 4D. Todo método contiene una o más líneas de instrucción. Cada línea de instrucción está compuesta de elementos del lenguaje.

Como usted ya ha trabajado en el Manual de Iniciación rápida, (ya lo ha hecho ¿verdad?) ya ha escrito y utilizado métodos.

Puede crear cinco tipos de métodos con 4D:

- **Métodos de objeto**: generalmente son métodos cortos utilizados para controlar objetos de formulario.
- **Métodos de formulario**: métodos que administran la visualización o impresión de un formulario.
- **Métodos de tabla/Triggers**: utilizados para aplicar las reglas de funcionamiento de su base.
- **Métodos de proyecto**: métodos disponibles para ser usados a través de su base. Por ejemplo, los métodos que pueden asociarse a menús.
- **Métodos base**: métodos utilizados para ejecutar inicializaciones o acciones especiales cuando abre o cierra una base, o cuando un navegador web se conecta a su base de datos publicada como un servidor web en Internet o Intranet.

Las siguientes secciones presentan cada tipo de método y explican brevemente la manera en que puede utilizar métodos para automatizar su base de datos.

Comenzar con los métodos de objeto

Todo objeto de un formulario puede realizar una acción (es decir, todo objeto activo) puede tener un método asociado. Un método de objeto monitorea y administra el objeto activo durante la entrada e impresión de datos. Un método de objeto está atado a su objeto activo incluso cuando el objeto es copiado y pegado. Esto le permite crear librerías de objetos activos reutilizables. El método de objeto toma control exactamente cuando es necesario.

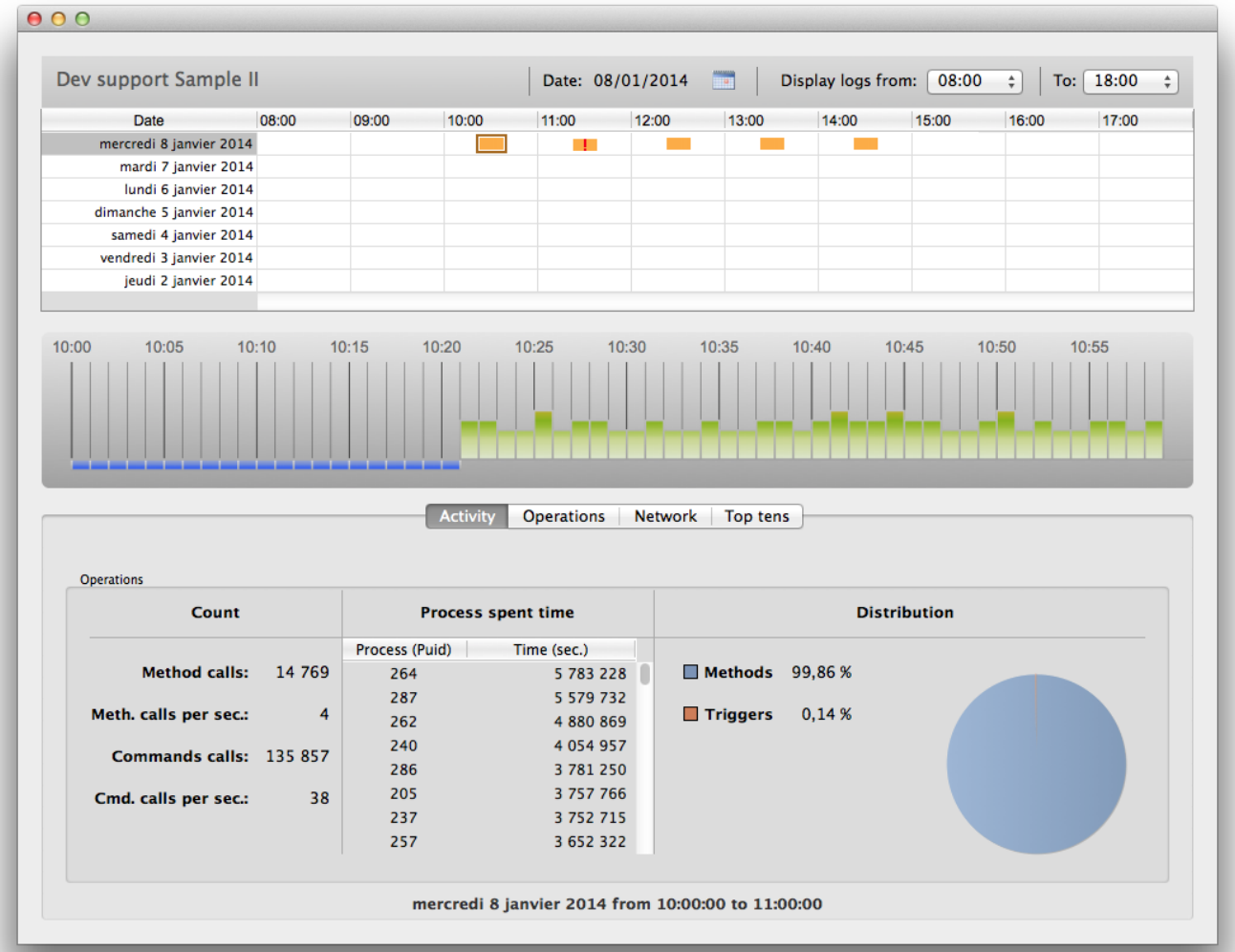
Los métodos de objeto son las herramientas primarias para la gestión de la interfaz del usuario, la cual es la entrada a su base de datos. La interfaz del usuario es el conjunto de procedimientos y convenciones por los cuales el ordenador se comunica con el usuario. La meta es hacer la interfaz del usuario de su base de datos tan simple y fácil de usar como sea posible. La interfaz del usuario debe hacer que la interacción con el ordenador sea un proceso agradable, que el usuario disfrute o incluso no note.

Hay dos tipos básicos de objetos activos en un formulario:

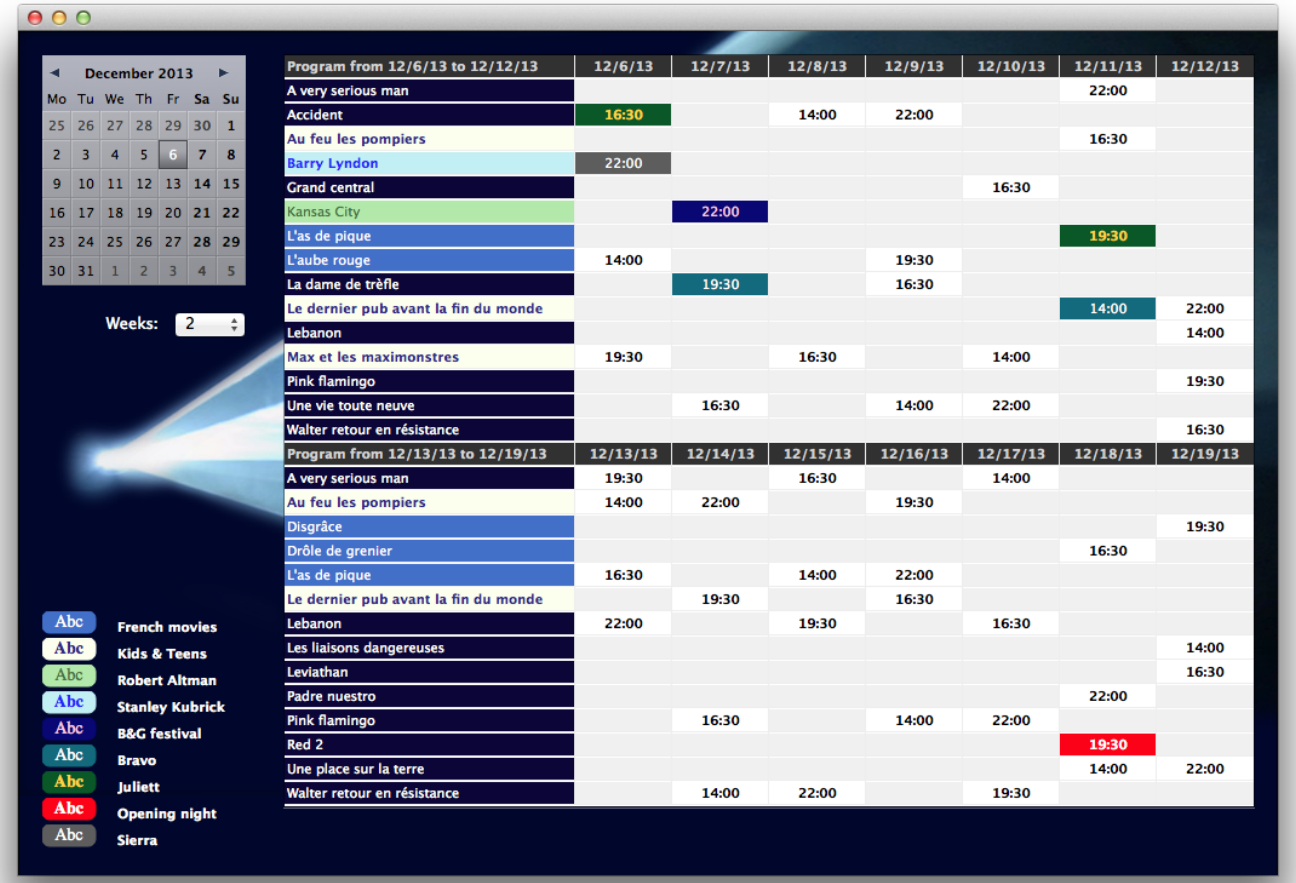
- Objetos activos para introducir, visualizar y guardar datos; tales como campos y subcampos
- Objetos activos de control; tales como áreas de entrada, botones, áreas de desplazamiento, listas jerárquicas, y termómetros.

4D le permite construir formularios clásicos, como este:

Igualmente puede construir formularios con múltiples controles gráficos, como este:



Incluso puede construir formularios que incorporen un estilo gráfico limitado sólo por su imaginación:



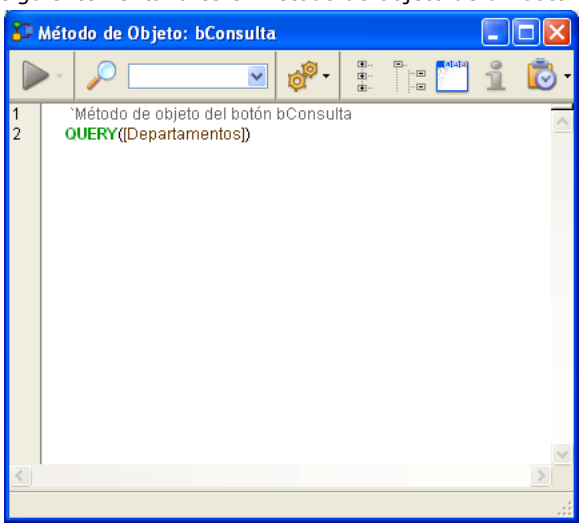
Program from 12/6/13 to 12/12/13	12/6/13	12/7/13	12/8/13	12/9/13	12/10/13	12/11/13	12/12/13
A very serious man						22:00	
Accident	16:30		14:00	22:00			
Au feu les pompiers						16:30	
Barry Lyndon	22:00						
Grand central					16:30		
Kansas City		22:00					
L'as de pique						19:30	
L'aube rouge	14:00			19:30			
La dame de trèfle		19:30		16:30			
Le dernier pub avant la fin du monde						14:00	22:00
Lebanon							14:00
Max et les maximonstres	19:30		16:30		14:00		
Pink flamingo							19:30
Une vie toute neuve		16:30		14:00	22:00		
Walter retour en résistance							16:30

Program from 12/13/13 to 12/19/13	12/13/13	12/14/13	12/15/13	12/16/13	12/17/13	12/18/13	12/19/13
A very serious man	19:30		16:30		14:00		
Au feu les pompiers	14:00	22:00		19:30			
Disgrâce							19:30
Drôle de grenier						16:30	
L'as de pique	16:30		14:00	22:00			
Le dernier pub avant la fin du monde		19:30		16:30			
Lebanon	22:00		19:30		16:30		
Les liaisons dangereuses							14:00
Leviathan							16:30
Padre nuestro						22:00	
Pink flamingo		16:30		14:00	22:00		
Red 2						19:30	
Une place sur la terre						14:00	22:00
Walter retour en résistance		14:00	22:00		19:30		

Sin importar cual sea su estilo de construir formularios, todos los objetos activos tienen ayudas integradas, como intervalos de valores y filtros para áreas de entrada, y acciones automáticas para controles, menús y botones. Utilice estas ayudas antes de añadir métodos de objeto. Las ayudas integradas son similares a los métodos en que permanecen asociadas con el objeto activo y se activan sólo cuando el objeto activo se utiliza. Generalmente, usted utilizará una combinación de ayudas integradas y de métodos de objeto para controlar la interfaz del usuario.

Generalmente, un método de objeto asociado a un objeto activo utilizado para entrada de datos realiza una tarea de gestión de datos específicamente para el campo o variable. El método puede realizar validación de datos, formato de datos, o cálculos. Incluso puede conseguir información relacionada con otros archivos. Algunas de estas tareas también pueden ejecutarse con las ayudas integradas de entrada de objetos. Utilice los métodos de objeto cuando la tarea a efectuar sea muy compleja para las ayudas integradas de entrada de datos. Para mayor información sobre ayudas integradas de entrada de datos, consulte el Manual de Diseño de 4D.

Los métodos de objeto también están asociados con objetos activos utilizados para control, como botones. Los objetos activos de control son esenciales para la navegación en su base. Los botones le permiten desplazarse de un registro a otro, cambiar de formulario, añadir y borrar datos. Estos objetos activos simplifican el uso de una base y reducen el tiempo de aprendizaje. Los botones también tienen ayudas integradas y, como para la entrada de datos, usted debe utilizar estas ayudas integradas antes de añadir métodos. Los métodos de objeto le permiten añadir acciones que no están integradas, a sus controles. Por ejemplo, la siguiente ventana es el método de objeto de un botón que cuando le hacen clic, muestra el editor de búsquedas.



A medida que se familiarice con los métodos de objeto, encontrará que puede crear librerías de objetos con métodos asociados. Puede copiar y pegar estos objetos y sus métodos en diferentes formularios, tablas y bases.

Controlar formularios con métodos de formulario

De la misma forma que los métodos de objeto están asociados a objetos activos en un formulario, un método de formulario está asociado a un formulario. Cada formulario puede tener un método de formulario. Un formulario es el medio a través del cual puede introducir, ver, e imprimir sus datos. Los formularios le permiten presentar los datos al usuario de diferentes formas. Gracias al uso de formularios, puede crear pantallas de entrada e informes impresos atractivos y fáciles de utilizar. Un método de formulario controla y administra el uso de un formulario específico, para la entrada e impresión de datos.

Los métodos de formulario administran formularios a un nivel más alto que los métodos de objeto. Los métodos de objeto se activan sólo cuando se utiliza el objeto, mientras un método de formulario se activa cuando se utiliza cualquier objeto en el formulario. Los métodos de formulario se utilizan generalmente para controlar la interacción entre los diferentes objetos y el formulario como un todo.

Como los formularios se utilizan de muchas maneras, es útil monitorear lo que pasa cuando su formulario está en uso. Para este propósito usted utiliza los diferentes **eventos de formulario**. Estos eventos le señalan lo que sucede actualmente con el formulario. Cada tipo de evento (por ejemplo, clics, doble-clic, pulsación de teclas...) activa o desactiva la ejecución del método de formulario como también el método de objeto de cada objeto del formulario.

Para mayor información sobre formularios, objetos, eventos y métodos, consulte la sección .

Regular el funcionamiento de su base de datos utilizando métodos de tabla/trigger

Un trigger está asociado a una tabla; por esta razón, también es llamado método de tabla. Los triggers se llaman automáticamente por el motor de base de datos 4D cada vez que manipula los registros de una tabla (Añadir, Borrar, Modificar y Cargar). Los triggers son métodos que pueden evitar que se realicen operaciones "ilegales" con los registros de su base de datos. Por ejemplo, en un sistema de facturación, puede evitar que un usuario añada una factura sin especificar el cliente a quien se le dirige. Los triggers son una herramienta poderosa para restringir operaciones en una tabla como también para prevenir pérdidas accidentales de datos o interferencias. Puede escribir triggers bastantes simples y luego volverlos más sofisticados.

Para mayor información sobre triggers, consulte la sección .

Utilizar métodos de proyecto en toda la base

A diferencia de los métodos de objeto, los métodos de formulario, y los triggers, los cuales están asociados con un objeto, formulario, o tabla en particular, los métodos de proyecto están disponibles para ser utilizados en toda su base. Los métodos de proyecto son reutilizables, y están disponibles para ser llamados por otros métodos. Si necesita repetir una tarea, no tiene que escribir métodos idénticos para cada caso. Puede llamar los métodos de proyecto desde donde los necesite, desde otros métodos de proyecto o desde métodos de objeto o de formulario. Cuando usted llama un método de proyecto, actúa como si hubiera escrito el método en la ubicación desde donde lo llamó. Los métodos de proyecto llamados por otros métodos son llamados "subrutinas."

Hay otra forma de utilizar los métodos de proyecto, asociándolos a los comandos de menús. Cuando asocia un método de proyecto a un comando de menú, el método se ejecuta cuando se selecciona el menú. Puede considerar el comando de menús como llamar a un método de proyecto.

Administrar las sesiones de trabajo con métodos de base de datos

De la misma forma que los métodos de objeto y de formulario son llamados cuando se producen eventos en un formulario, hay métodos asociados a la base los cuales son llamados cuando se produce un evento de sesión de trabajo. Esto son los **métodos de base de datos**. Por ejemplo, cada vez que abra una base, puede querer inicializar algunas variables que utilizará durante la sesión de trabajo. Para hacer esto, utilice el método base **On Startup**, que se ejecuta automáticamente por 4D cuando abre la base.

Para mayor información sobre Métodos base, consulte la sección .

Desarrollar su base de datos

El desarrollo es proceso de personalización de una base utilizando el lenguaje y otras herramientas integradas.

Simplemente creando una base de datos, usted ya ha dado los primeros pasos para utilizar el lenguaje. Todas las partes de su base, las tablas y los campos, los formularios y sus objetos y los menús, están relacionadas con el lenguaje. El lenguaje de 4D "conoce" sobre todas estas partes de su base.

Tal vez su primer uso del lenguaje sea añadir un método a un objeto de formulario para controlar la entrada de datos. Luego, puede añadir un método de formulario para controlar la visualización de su formulario. A medida que la base de datos se vuelve más compleja, puede añadir una barra de menú con métodos de proyecto para personalizar completamente su base.

Como todos los otros aspectos de 4D, el desarrollo es un proceso muy flexible. No hay una ruta formal a tomar durante el desarrollo, puede desarrollar de la manera en la que se sienta a gusto. Por supuesto, hay algunas pautas en el proceso.

- Implementación: implementación de su diseño en el entorno Diseño.
- Pruebas: se pone a prueba el diseño y prueba cada elemento personalizado utilizando el comando Test Aplication para incializar el entorno de Aplicación.
- Uso: cuando su base se personaliza completamente, utilícela en el entorno Aplicación.
- Correcciones: si encuentra errores, vuelva al entorno Diseño para corregirlos.

Las herramientas especiales de soporte al desarrollo, ocultas hasta que sean necesarias, están integradas a 4D. A medida que utilice el lenguaje con más frecuencia, encontrará que estas herramientas facilitan el proceso de desarrollo. Por ejemplo, el editor de métodos encuentra los errores de digitación y le da formato a su trabajo; el interprete (el motor que ejecuta el lenguaje) encuentra errores de sintaxis y se los señala; y el Depurador le permite monitorear la ejecución de sus métodos para detectar errores de diseño.

Construir aplicaciones

Por el momento usted está familiarizado con los usos generales de una base de datos, entrada de datos, búsquedas, ordenación y creación de informes. Ha realizado estas tareas en el entorno Diseño, utilizando los menús y editores integrados.

Cuando utiliza una base de datos, repite ciertas secuencias de tareas. Por ejemplo, en una base de datos de contactos personales, podría buscar colegas, ordenarlos por apellido, e imprimir un informe específico cada vez que se modifique la información. Estas tareas no parecen difíciles de realizar, pero pueden tardar bastante cuando hay que hacerlas 20 veces. Además, si no utilizó la base un par de semanas, puede que haya olvidado algunos de los pasos para generar informes. Los

pasos en métodos están encadenados unos a otros, de manera que un sólo comando automáticamente realiza todas las tareas asociadas a él. Por lo tanto, no tiene que preocuparse por pasos específicos.

Las aplicaciones tienen menús personalizados y realizan tareas que son específicas para las necesidades de la persona que utiliza su base. Una aplicación está compuesta de todos los elementos de su base: la estructura, los formularios, los métodos de objeto, de formulario y de proyecto, los menús, y las contraseñas.
Puede compilar sus bases y crear aplicaciones Windows y Macintosh autónomas. La compilación de bases de datos aumenta la velocidad de ejecución del lenguaje, protege sus bases de datos y le permite crear aplicaciones que son completamente independientes. El compilador integrado también verifica la sintaxis y los tipos de variables en métodos por coherencia.

Una aplicación puede ser tan simple como un menú sencillo que le permite introducir los nombres de personas e imprimir un informe, o tan compleja como un sistema de facturación, inventario, y control. Los usos de las aplicaciones son ilimitados. Generalmente, una aplicación crece de una base de datos utilizada en el entorno Diseño a una base de datos controlada completamente por menús personalizados y formularios.

¿A dónde ir desde aquí?

• El desarrollo de aplicaciones puede ser tan simple o tan complejo como quiera. Para mayor información sobre la construcción de una aplicación 4D sencilla, consulte la sección .
• Si es nuevo con 4D, consulte las sección para aprender sobre los principios básicos del lenguaje 4D: comience con .

Construir una aplicación 4D

Una aplicación es una base de datos diseñada para responder a una necesidad específica. Tiene una interfaz de usuario diseñada especialmente para facilitar su uso. Las tareas que una aplicación realiza están limitadas a aquellas apropiadas para su objetivo. La creación de aplicaciones con 4D es más fácil que con programación tradicional. 4D se puede utilizar para crear una variedad de aplicaciones, incluyendo:

- Un sistema de facturación
- Un sistema de control de inventarios
- Un sistema de contabilidad
- Un sistema de pago de nómina
- Un sistema de recursos humanos
- Un sistema de seguimiento de clientes
- Una base de datos accesible por Internet o Intranet

Incluso es posible que una sola aplicación pueda contener todos estos sistemas. Este tipo de aplicaciones corresponden al uso tradicional de bases de datos. Adicionalmente, las herramientas en 4D le permiten crear aplicaciones originales, tales como:

- un sistema de seguimiento de documentos
- un sistema gráfico de manejo de imágenes
- una aplicación de publicación de catálogos
- un sistema de control y monitoreo de un dispositivo serial
- un sistema de mensajería electrónica (E-mail)
- un sistema de programación multiusuario
- una lista como una lista de menús, una colección de videos, o una colección de música

Generalmente, una aplicación puede comenzar como una base de datos utilizada en el entorno Diseño. La base se "transforma" en una aplicación a medida que se personaliza. Lo que diferencia una aplicación es que el sistema necesario para manejar la base de datos está oculto del usuario. La gestión de la base está automatizada, y los usuarios utilizan menús para realizar tareas específicas.

Cuando utiliza una base 4D en el entorno Diseño, debe conocer los pasos a seguir para obtener un resultado. En una aplicación 4D, se utiliza el modo Aplicación, en el cual debe manejar todos los aspectos que son automáticos en el entorno Diseño, es decir:

- Navegación entre tablas: la ventana **Lista de tablas**, el comando **Últimas tablas utilizadas** o los botones de navegación no están disponibles para el usuario. Puede utilizar los comandos de menús y métodos para controlar la navegación entre tablas.
- Menús: en modo Aplicación, sólo aparecen por defecto los menús Archivo con el comando de menú Salir, Edición, Modo y Ayuda (así como el menú aplicación bajo Mac OS). Si la aplicación necesita más menús, debe crearlos y manejarlos utilizando métodos 4D o acciones estándar.
- Editores: los editores, como los editores de búsqueda y de ordenación, no están disponibles automáticamente en el entorno Aplicación. Si quiere utilizarlos, debe llamarlos utilizando métodos 4D.

Las siguientes secciones incluyen ejemplos de automatización del uso de una base utilizando el lenguaje.

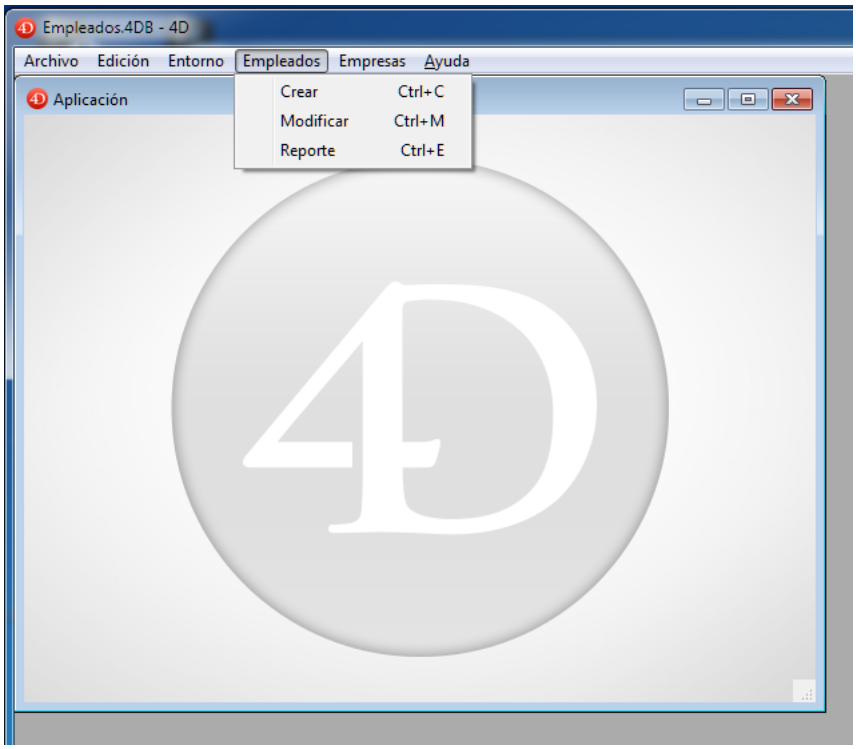
Entorno Aplicación: un ejemplo

Los menús personalizados son la interfaz primaria de una aplicación. Los menús personalizados facilitan a los usuarios el aprendizaje y utilización de una base de datos. Crear menús personalizados es muy sencillo, asocie métodos o acciones estándar a cada comando de menú (también llamados elementos de menús) en el editor de menús.

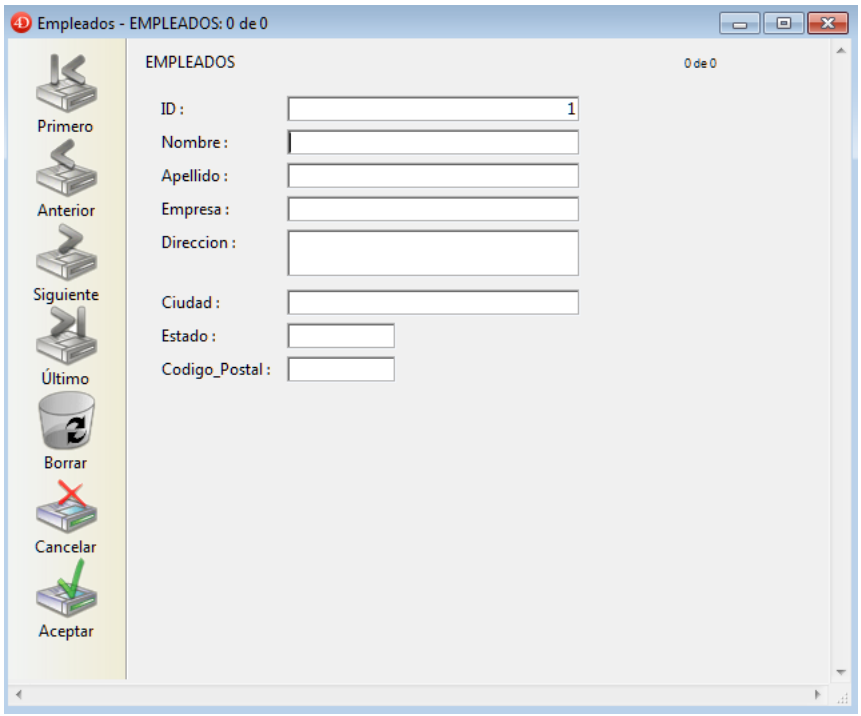
La sección "Punto de vista del usuario" describe lo que sucede cuando el usuario selecciona un comando de menú. Luego, "Detrás del escenario" describe el trabajo de diseño que se ha efectuado. Aunque el ejemplo es sencillo, es evidente cómo los menús personalizados hacen que una base sea más fácil para aprender y utilizar. En el entorno Diseño, en lugar de las herramientas "genéricas" y comandos de menú, el usuario sólo ve elementos que corresponden a sus necesidades.

Punto de vista del usuario

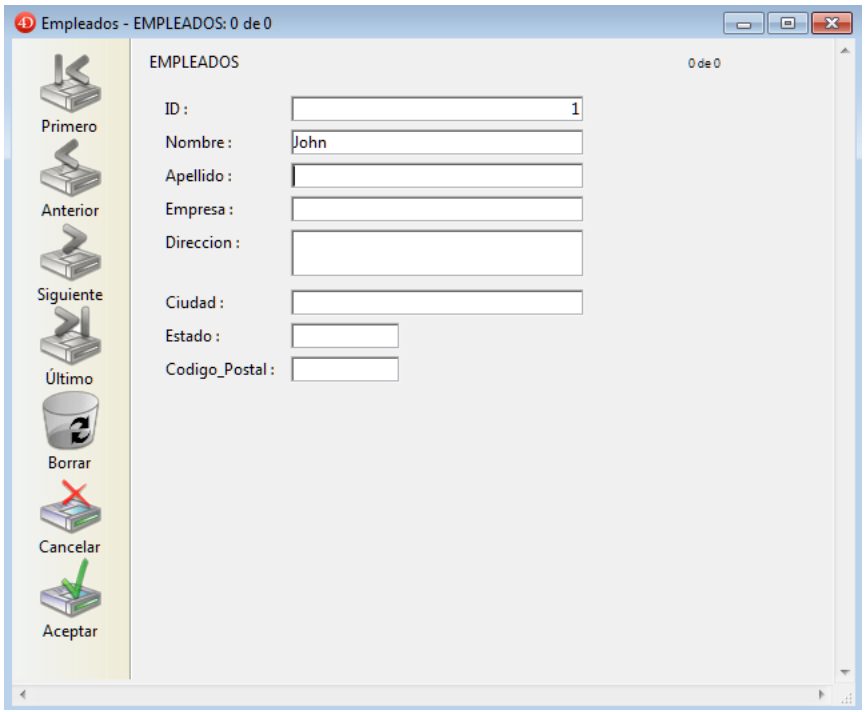
El usuario selecciona un comando de menú llamado **Crear** del menú **Personas** para añadir una nueva persona a la base de datos.



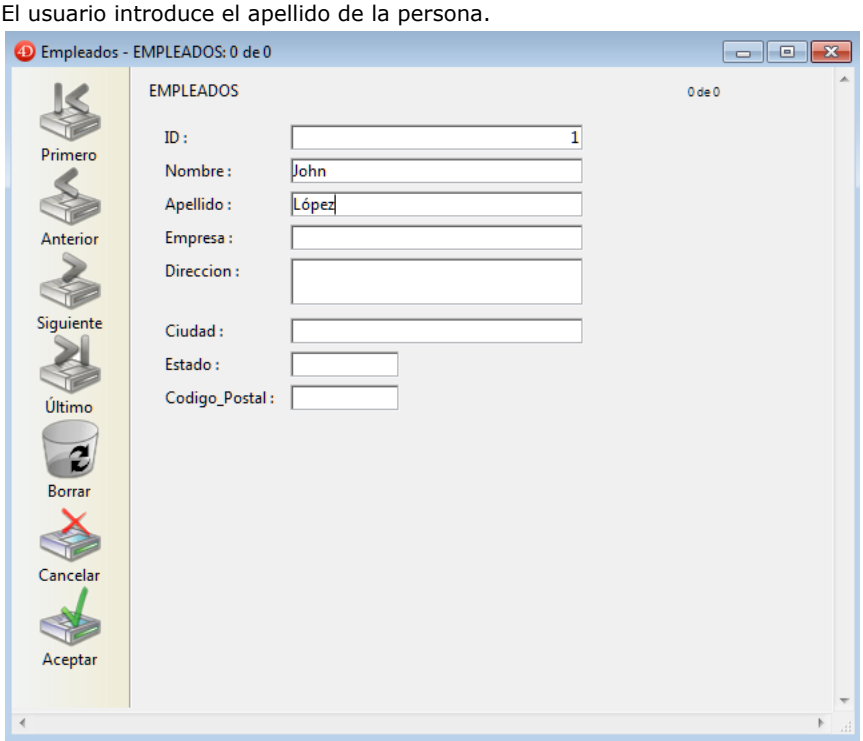
Aparece el formulario de entrada de la tabla Personas.



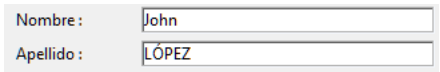
El usuario introduce el nombre de la persona y luego presiona la tecla Tab para pasar al siguiente campo.



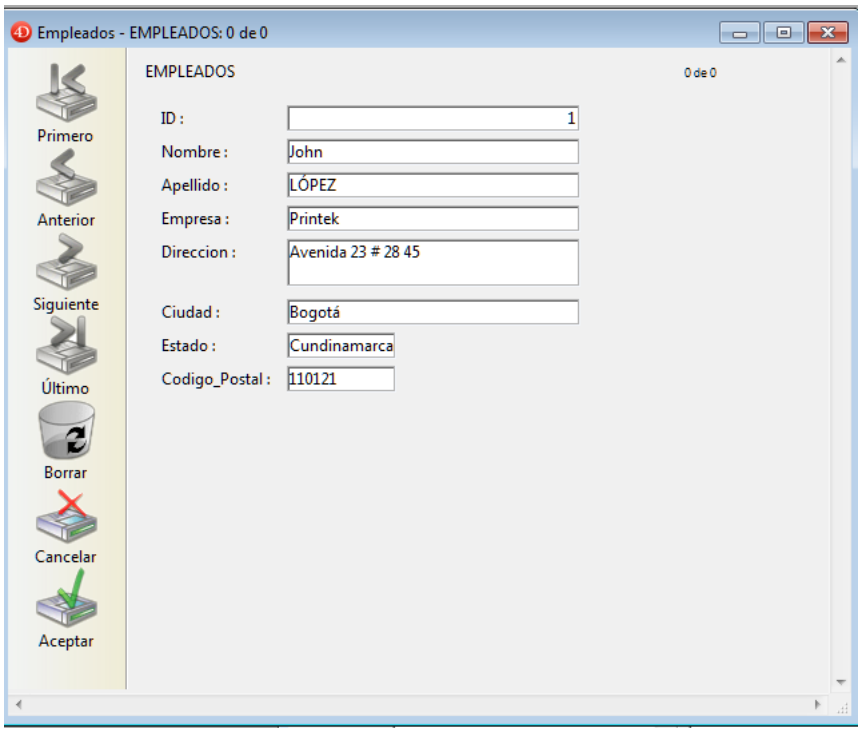
El usuario introduce el apellido de la persona.



El usuario presiona la tecla Tab para pasar al siguiente campo: el apellido se convierte en letras mayúsculas.



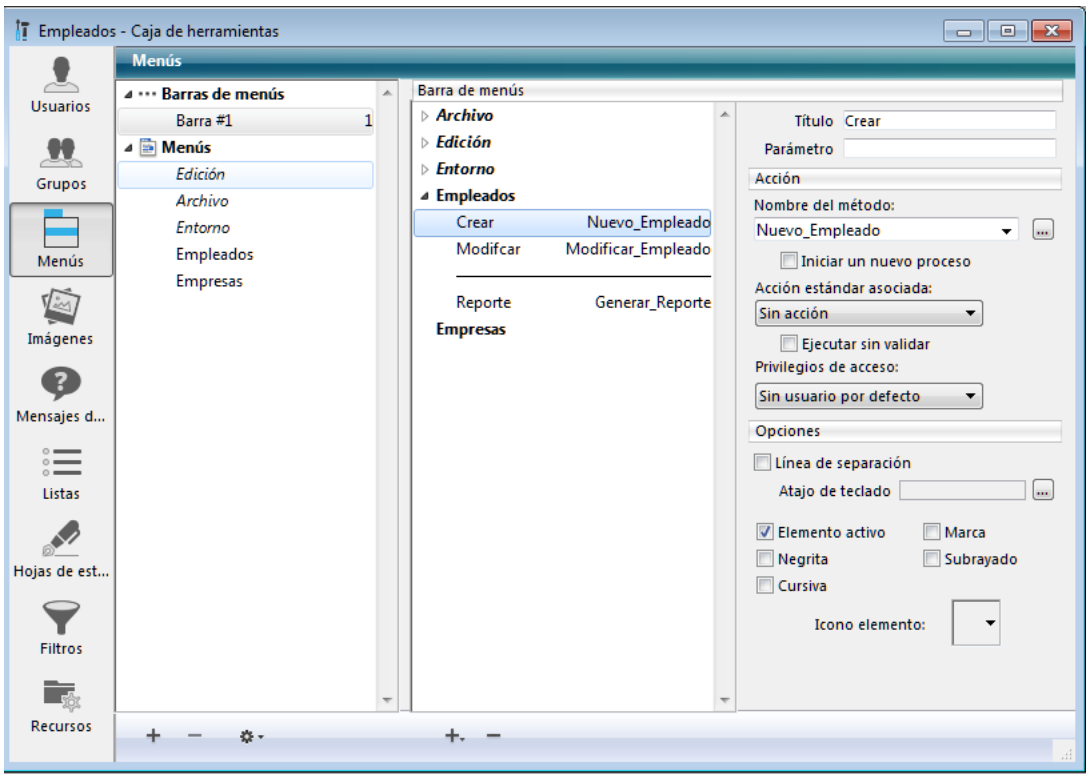
Una vez el usuario termina de introducir el registro y hace clic en el botón de validación (generalmente el último botón en la barra de botones).



Aparece otro registro vacío, y el usuario hace clic en el botón Cancelar (el que tiene una "X") para terminar el "bucle de entrada de datos." El usuario vuelve a la barra de menús.

Detrás del escenario

La barra de menús fue creada en el entorno Diseño, utilizando el editor de menús.



El comando de menú **Nuevo** está asociado a un método de proyecto llamado ***Nueva persona***. Este método fue creado en el entorno Diseño, utilizando el editor de métodos.

Cuando el usuario selecciona este comando de menú, el método ***Nueva persona*** se ejecuta:

```
Repeat
   ADD RECORD([Personas])
Until(OK=0)
```

El bucle **Repeat...Until** al interior del cual se encuentra el comando **ADD RECORD** tiene exactamente los mismo efectos que le comando de menú **Nuevo Registro** en el entorno Diseño. Muestra el formulario de entrada al usuario, de manera que pueda añadir un nuevo registro. Cuando el usuario guarda el registro, aparece otro nuevo registro vacío. Este bucle **ADD RECORD** continúa ejecutándose hasta que el usuario hace clic en el botón Cancelar.

Cuando se introduce un registro, ocurre lo siguiente:

- No hay método para el campo **Nombre**, entonces no se ejecuta nada.

- Hay un método para el campo **Apellido**. Este método fue creado en el entorno Diseño utilizando los editores de formularios y métodos. El método se ejecuta:

```
[Personas]Apellido:=Uppercase([Personas]Apellido)
```

Esta línea convierte los caracteres del campo **Apellido** en mayúsculas.

Después de introducir un registro, cuando el usuario hace clic en el botón Cancelar en el formulario siguiente, la variable OK toma el valor cero, terminando la ejecución del bucle **ADD RECORD**.

Como no hay más instrucciones a ejecutar, el método **Nueva persona** se detiene y devuelve el control a la barra de menús.

Comparar una aplicación 4D con el entorno Diseño

Comparemos la manera en que una tarea se realiza en el entorno Diseño y la misma tarea realizada utilizando el lenguaje. La tarea es:

- Encontrar un grupo de registros
- Ordenarlos
- Imprimir un informe

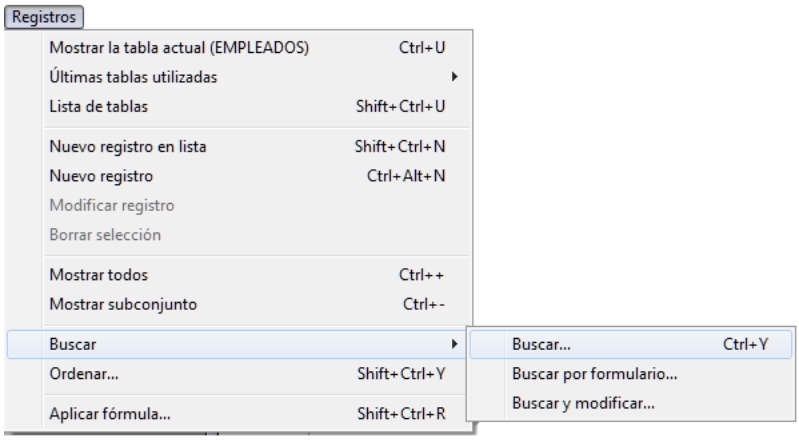
La próxima sección, "Utilizando una base en el entorno Diseño," muestra las tareas realizadas en el entorno Diseño.

La siguiente sección, "Utilizando los editores integrados en el entorno Aplicación," muestra las mismas tareas realizadas en una aplicación.

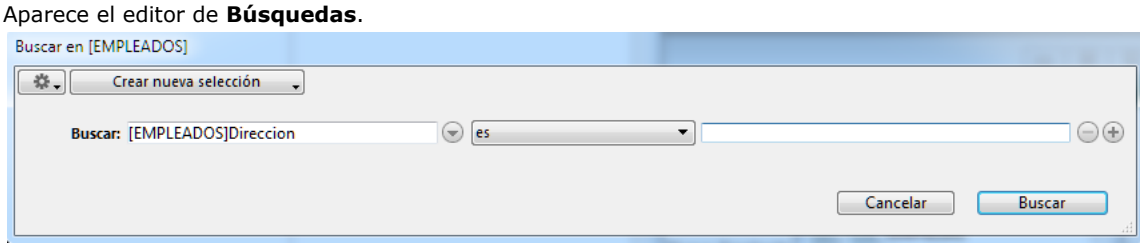
Note que aunque ambos métodos realizan la misma tarea, los pasos en la segunda sección se automatizan por programación.

Utilizando una base en el entorno Diseño

El usuario selecciona **Buscar**>**Buscar...** en el menú **Registros**.

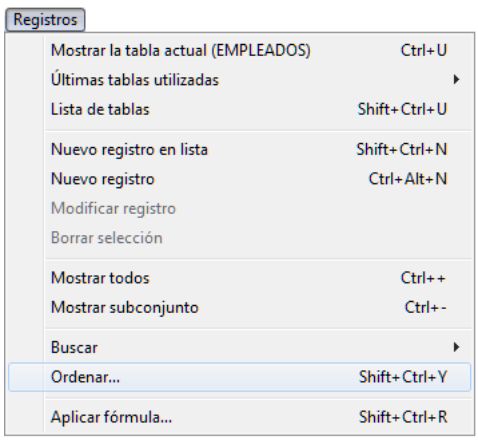


Aparece el editor de **Búsquedas**.

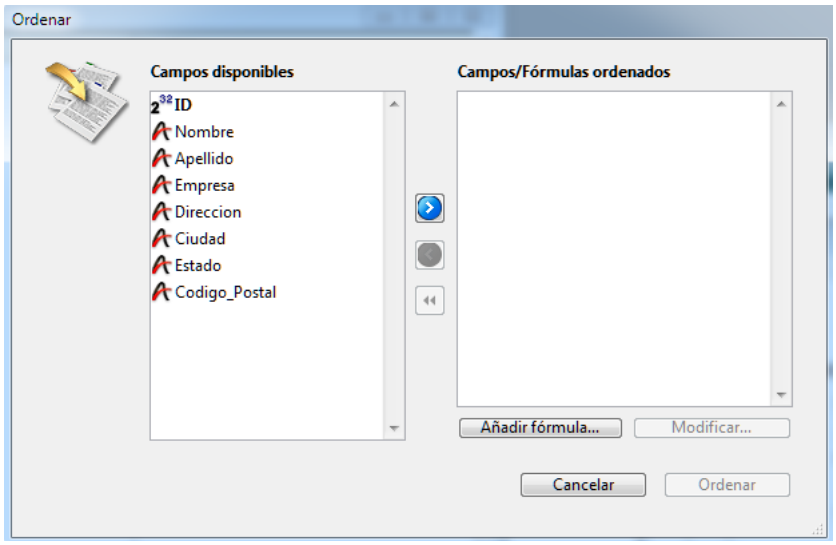


El usuario introduce el criterio de búsqueda y hace clic en el botón **Buscar**. Se efectúa la búsqueda.

El usuario selecciona **Ordenar...** en el menú **Registros**.



Aparece el editor **Ordenar**.



El usuario define sus criterios de búsqueda y hace clic en el botón **Ordenar**. Se efectúa la ordenación.

Luego, para imprimir los registros, son necesarios los siguientes pasos:

- El usuario selecciona **Imprimir** en el menú **Archivo**.
- Aparece la caja de diálogo para seleccionar el formulario a imprimir, porque los usuarios necesitan saber qué formulario imprimir.
- Aparecen las cajas de diálogo de impresión. El usuario elige los parámetros, y se imprime el informe.

Usar los editores integrados en el entorno Aplicación

Examinemos cómo estas operaciones pueden efectuarse en el entorno Aplicación.

El usuario selecciona **Informe** en el menú **Personas**.

Incluso en este punto, utilizar una aplicación es más fácil para los usuarios, ellos no necesitan saber que el primer paso es efectuar una búsqueda.

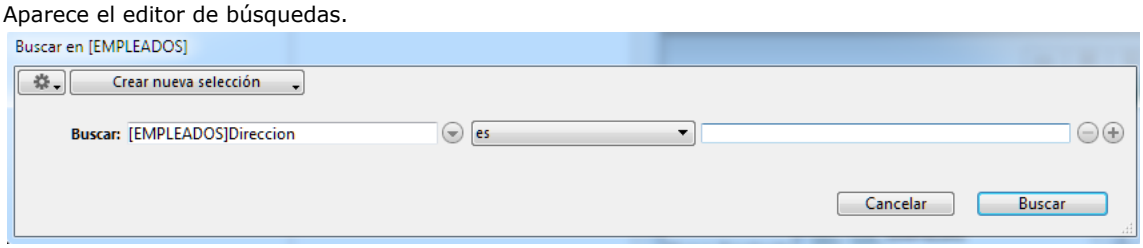
Un método llamado **Build Report** se asocia al comando de menú; se ve de esta forma:

```
QUERY([Employees])
ORDER BY([Employees])
FORM SET OUTPUT([Employees];"Report")
PRINT SELECTION([Employees])
```

Se ejecuta la primera línea:

```
QUERY([Employees])
```

Aparece el editor de búsquedas.



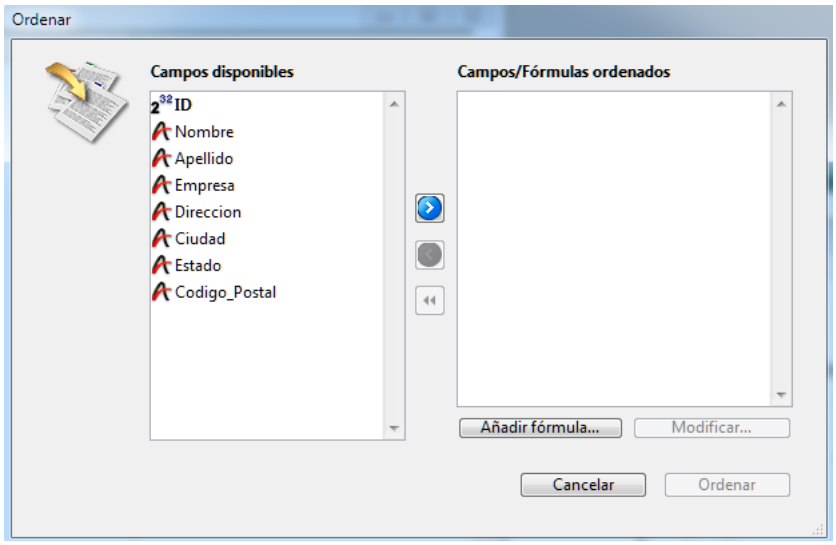
Los usuarios introducen el criterio de búsqueda y hacen clic en el botón **Buscar**. Se efectúa la búsqueda.

Se ejecuta la segunda línea del método **Build Report**:

```
ORDER BY([Employees])
```

Observe que el usuario no necesita saber que ordenar los registros es el siguiente paso.

Aparece el editor **Ordenar**.



El usuario introduce los criterios de búsqueda y hace clic en el botón **Ordenar**. Se efectúa la ordenación.

Se ejecuta la tercera línea del método **Build Report**:

```
FORM SET OUTPUT([Employees];"Report")
```

Una vez más, el usuario no necesita saber que hacer después; el método se encarga de eso.

Se ejecuta la línea final del método **Build Report**:

```
PRINT SELECTION([Employees])
```

Aparecen las cajas de diálogo de impresión. El usuario fija los parámetros y se imprime el informe.

Automatizar la aplicación aún más

Los mismos comandos utilizados en el ejemplo anterior pueden usarse para automatizar más la base de datos.

Miremos la nueva versión del método **Build Report**.

El usuario elige **Informe** en el menú **Personas**. Un método llamado **Build Report2** se asocia al comando de menú. Se ve de esta manera:

```
QUERY([Employees];[Employees]Company="Acme")
ORDER BY([Employees];[Employees]Last Name;>;[People]First Name;>)
FORM SET OUTPUT[Employees];"Report")
PRINT SELECTION([Employees];*)
```

Se ejecuta la primera línea:

```
QUERY([Employees];[Employees]Company="Acme")
```

No aparece el editor de búsquedas. En su lugar, la consulta se especifica y realiza por el comando **QUERY**. El usuario no tiene que hacer nada.

Se ejecuta la segunda línea del método **Build Report2**:

```
ORDER BY([Employees];[Employees]Last Name;>;[People]First Name;>)
```

El editor **Ordenar** no aparece, y la ordenación se realiza de inmediato. Una vez más, no es necesaria ninguna acción por parte del usuario.

Se ejecutan las líneas finales del método **Build Report2**:

```
FORM SET OUTPUT[Employees];"Report")
PRINT SELECTION([Employees];*)
```

Las cajas de diálogo de impresión no aparecen. El comando **PRINT SELECTION** acepta un parámetro opcional asterisco (*) que le indica al comando que utilice los parámetros de impresión vigentes cuando el formulario del informe fue creado. Se imprime el informe.

Esta automatización adicional le ahorra al usuario tener que introducir opciones en tres cajas de diálogo. Estos son los beneficios:

- La búsqueda se efectúa automáticamente: esto evita que los usuarios seleccionen un criterio de búsqueda equivocado cuando hacen una búsqueda.
- La ordenación se realiza automáticamente: esto evita que los usuarios seleccionen criterios de ordenación equivocados cuando definen una ordenación.
- La impresión se efectúa automáticamente: esto evita que los usuarios seleccionen un formulario equivocado para imprimir.

Ayudas para el desarrollo de aplicaciones 4D

A medida que desarrolla una aplicación 4D, descubrirá muchas funcionalidades que no había observado cuando comenzó. Incluso puede aumentar la versión estándar de 4D añadiendo otras herramientas y plug-ins a su entorno de desarrollo 4D.

Plug-ins 4D

4D ofrece varios plug-ins que pueden utilizarse para aumentar las capacidades de sus aplicaciones.

- **4D Write**: procesador de palabras
- **4D View**: hoja de cálculo y editor de listas
- **4D Internet Commands** (integrado): utilidades de comunicación vía Internet.

- **4D ODBC Pro**: conectividad vía ODBC
- **4D for OCI**: conectividad con ORACLE Call Interface

Para mayor información, contacte a 4D o visite nuestro sitio web:

http://www.4d.com

La comunidad 4D y herramientas de terceras partes

Existe una comunidad 4D internacional muy activa, compuesta por grupos de usuarios, foros electrónicos y Partners 4D. Los Partners 4D producen **Herramientas de terceras partes**. Puede suscribirse al foro de usuarios de 4D en la siguiente dirección:
http://forums.4D.fr

La comunidad 4D ofrece acceso consejos y soluciones, información y herramientas adicionales que le ahorrarán tiempo y energía y aumentarán su productividad.

Presentación del lenguaje

- Introducción al lenguaje 4D
- Tipos de datos
- Constantes
- Variables
- Variables sistema
- Punteros
- Convenciones
- Condiciones y bucles
- If...Else...End if
- Case of...Else...End case
- While...End while
- Repeat...Until
- For...End for
- For each...End for each
- Use...End use
- Métodos
- Métodos de proyecto

Introducción al lenguaje 4D

El lenguaje 4D está formado por diferentes componentes, que le permiten realizar tareas y administrar sus datos.

- **Tipos de datos**: categorías de datos en una base. Vea la descripción en esta sección y una descripción detallada en la sección **Tipos de datos**.
- **Variables**: lugares temporales de almacenamiento de datos en memoria. Para una descripción detallada consulte la sección **Variables**.
- **Operadores**: símbolos que realizan un cálculo entre dos valores. Vea la descripción en esta sección y una descripción detallada en la sección **Operadores** y sus subsecciones.
- **Expresiones**: combinaciones de otros componentes que resultan en un valor. Vea la descripción en esta sección.
- **Comandos**: instrucciones integradas para realizar una acción. Todos los comandos de 4D, tal como **ADD RECORD**, son descritos en este manual, agrupados por tema; cuando es necesario, el tema es precedido por una sección de introducción. Puede utilizar los plug-ins 4D para añadir nuevos comandos a su entorno de desarrollo 4D. Por ejemplo, una vez instalado el plug-in 4D Write en su sistema 4D, los comandos 4D Write quedan disponibles para la creación y manipulación de los documentos de procesamiento de palabras.
- **Constantes predefinidas**: valores constantes accesibles por nombre. Por ejemplo, XML DATA es una constante (valor 6). Las constantes predefinidas permiten escribir código más legible. Las constantes se describen con los comandos que los utilizan y están totalmente listadas en la sección **Lista de temas de constantes**.
- **Métodos**: instrucciones que usted escribe utilizando todas las parte del lenguaje descritas aquí. Vea la descripción en la sección **Métodos** y sus subsecciones.

Esta sección presenta los **Tipos de datos**, **Operadores** y **Expresiones**. Para una descripción de los otros componentes, consulte las secciones mencionadas anteriormente.

Adicionalmente:

- Elementos del lenguaje como variables, tienen nombres llamados Identificadores. Para una descripción detallada sobre identificadores y reglas para nombrar objetos, consulte la sección **Convenciones**.
- Para aprender más sobre variables Array, consulte la sección **Arrays**.
- Si tiene la intención de compilar su base, consulte la sección **Comandos del Compilador** así como el Manual de Diseño de 4D.

Lenguaje para los comandos y constantes

A partir de 4D v15, el editor de métodos de 4D utiliza por defecto el lenguaje internacional "Inglés-US", independientemente de la versión 4D o de la configuración del sistema local. Esta funcionalidad neutraliza las variaciones regionales que puedan afectar la interpretación del código entre aplicaciones 4D (formatos de fecha, por ejemplo); y en las versiones francesas de 4D, los comandos y las constantes ahora se escriben en "Inglés-US" como ya ocurre en otros idiomas.

Esta configuración predeterminada ofrece a los desarrolladores 4D varias ventajas:

- Facilita el intercambio de código entre los desarrolladores, con independencia de su país, configuración regional, o de la versión de 4D utilizada. Un método 4D ahora se puede cambiar mediante una simple copiar/pegar, o guardar en un archivo de texto, sin problemas de compatibilidad.
- También hace posible la inclusión de métodos 4D en herramientas de control de fuentes, que a menudo requieren que las exportaciones sean independientes de los parámetros regionales y de los lenguajes.

Este ajuste se puede desactivar mediante la opción "Usar configuración del sistema regional" en el diálogo Preferencias de 4D (ver **Página Métodos**).

Principios de entrada en lenguaje English-US

La configuración Inglés-US pueden tener varios efectos sobre la forma de escribir métodos. Esto concierne al código en modo de desarrollo, así como también a las fórmulas en las aplicaciones finales. En este modo, el código debe cumplir con las normas siguientes:

- El separador decimal para números reales deben ser ahora el punto (".") en todas las versiones (y no comas (",") como es la costumbre en francés, por ejemplo).
- Las constantes de tipo fecha ahora deben utilizar el formato ISO (!AAAA-MM-DD!) en todas las versiones.
- los nombres de comandos y de constantes deben estar en Inglés (este cambio sólo se refiere a las versiones francesas de 4D, ya que esto ya era el caso de otros lenguajes).

Nota: el editor de métodos incluye mecanismos específicos que corrigen automáticamente entradas incorrectas en caso de necesidad.

La siguiente tabla ilustra las diferencias entre el código en 4D v15 (o superior) y las versiones anteriores:

	Ejemplos de código en los métodos/fórmulas
4D v15 y superior (modo por defecto, todas las versiones)	a:=12.50 b:=!2013-12-31! Current date
4D v14 o 4D v15 (preferencia seleccionada, versión US, por ejemplo)	a:=12.50 b:=!12/31/2013! Current date
4D v14 o 4D v15 (preferencia seleccionada, versión FR, por ejemplo)	a:=12,50 b:=!31/12/2013! Date du jour

Nota: cuando la preferencia está seleccionada, los formatos de fecha y real se basan en los parámetros sistema.

Tipos de datos

En el lenguaje, los diferentes tipos de datos que se pueden majejar son llamados tipos de datos. Hay tipos de datos básicos (cadena, numérico, fecha, hora, Booleano, imagen, punteros, arrays), y también tipos de datos compuestos (BLOBs, objetos, colecciones).

Observe que en la lista de tipos de datos, los tipos de datos cadena y numérico están asociados a más de un tipo de campo. Cuando los datos son ubicados dentro de un campo, el lenguaje convierte automáticamente los datos al tipo correcto para el campo. Por Ejemplo, si se utiliza un campo entero, los valores que contiene se tratan automáticamente como numéricos. En otras palabras, no debe preocuparse por mezclar campos de tipos similares cuando utilice el lenguaje; él lo hará por usted.

Sin embargo, es importante cuando utiliza el lenguaje, no mezclar diferentes tipos de datos. De la misma forma que no tiene sentido almacenar "ABC" en un campo Fecha, no tiene sentido colocar "ABC" en una variable utilizada para fechas. En la mayoría de los casos, 4D es muy tolerante y tratará de utilizar de manera lógico lo que usted está haciendo. Por ejemplo, si añade un número a una fecha, 4D asumirá que quiere añadir este número de días a la fecha, pero si trata de añadir una cadena a una fecha, 4D le dirá que la operación no es posible.

Hay casos en los cuales usted necesita almacenar datos de un tipo y utilizar otro tipo. El lenguaje contiene un conjunto de comandos que le permiten convertir tipos de datos. Por ejemplo, si quiere crear un número de matricula que comience con un número y termine con caracteres como "abc". En este caso, podría escribir:

```
[Productos]Matricula:=String(Número)+"abc"
```

Si *Numero* es 17, entonces *[Productos]Matricula* tendrá la cadena "17abc".

Los tipos de datos se describen en detalle en la sección **Tipos de datos**.

Operadores

Cuando utiliza el lenguaje, es raro que sólo quiera datos. Es más probable que quiera hacer algo con los datos. Usted realiza cálculos con los operadores. Los operadores, en general, toman dos valores y realizan una operación que da como resultado un nuevo valor. Usted ya está familiarizado con muchos operadores. Por ejemplo, 1 + 2 utiliza el operador de adición (o signo más) para sumar dos números, y el resultado es 3. Esta tabla muestra algunos operadores numéricos familiares:

Operador	Operación	Ejemplo
+	Suma	1 + 2 es igual a 3
–	Resta	3 – 2 es igual a 1
*	Multiplicación	2 * 3 es igual a 6
/	División	6 / 2 es igual a 3

Los operadores numéricos son sólo un tipo de operador disponible. 4D soporta múltiples tipos de datos, tales como números, texto, fecha, e imágenes, de manera que hay operadores que realizan operaciones en estos diferentes tipos de datos.

Con frecuencia se utilizan los mismos símbolos para diferentes operaciones, dependiendo del tipo de datos. Por ejemplo, el signo (+) realiza diferentes operaciones con diferentes datos:

Tipo de datos	Operación	Ejemplo
Número	Suma	1 + 2 suma los números y el resultado es 3
Cadena	Concatenación	"Buenos" + "días" concatena (une) las cadenas y el resultado es "Buenos días"
Fecha y número	Adición de fecha	!1997/1/1! + 20 añade 20 días a la fecha Enero 1, 1989, y el resultado es la fecha 21 de enero, 1989

Los operadores son definidos en detalle en la sección **Operadores** y sus subsecciones.

Expresiones

En palabras sencillas, las expresiones devuelven un valor. De hecho, cuando utiliza el lenguaje 4D, utiliza todo el tiempo expresiones y tiende a pensar en ellas sólo en términos del valor que representan. Las expresiones también son llamadas fórmulas.

Las expresiones están constituidas de casi todas las otras partes del lenguaje: comandos, operadores, variables, y campos. Usted utiliza expresiones para construir líneas de código, que a su vez se utilizan para construir métodos. El lenguaje utiliza expresiones cada vez que necesita conocer el valor de un dato.

Rara vez las expresiones son “independientes.” Sólo hay unos pocos lugares en 4D donde una expresión puede ser utilizada como tal:

- en las cajas de diálogo de búsqueda por fórmula
- en el depurador donde el valor de las expresiones puede ser evaluado
- en la caja de diálogo aplicar fórmula
- en el editor de informes rápidos como una fórmula en una columna

Una expresión puede ser simplemente una constante, tal como el número 4 o la cadena “Hola.” Como su nombre lo indica, el valor de una constante nunca cambia. Cuando las expresiones contienen operadores comienzan a volverse interesantes. En las secciones anteriores usted vio expresiones que utilizan operadores. Por ejemplo, 4 + 2 es una expresión que utiliza el operador de adición para sumar dos números y cuyo resultado es 6.

Usted se refiere a una expresión por el tipo de dato que devuelve. Hay varios tipos de expresiones:

- Expresión cadena
- Expresión numérica (también llamada número)
- Expresión fecha
- Expresión hora
- Expresión Booleana
- Expresión Imagen
- Expresión Puntero
- Expresión Objeto
- Expresión Colección.

La tabla siguiente da un ejemplos para cada tipo de expresión.

Expresión	Tipo	Explicación
"Buenos"	Cadena	La palabra Buenos es una cadena constante, indicado por las comillas.
"Buenos " + "días"	Cadena	Dos cadenas, "Buenos " y "días", se unen (concatenan) con la ayuda del operador de concatenación (+). Se devuelve la cadena "Buenos días".
"Sr." + [Personas]Nombre	Cadena	Dos cadenas se concatenan: la cadena "Sr." y el valor actual del campo Nombre en la tabla Personas. Si el campo contiene "López", la expresión devuelve "Sr. López".
Uppercase ("Suárez")	Cadena	Esta expresión utiliza **Uppercase**, un comando del lenguaje, para convertir la cadena "suárez" a mayúsculas. un comando del lenguaje, para convertir la cadena "Suárez" a mayúsculas. Devuelve "SUÁREZ".
4	Númerico	Esta es una constante numérica, 4.
4 * 2	Númerico	Dos números, 4 y 2, se multiplican utilizando el operador de multiplicación (*). El resultado es el número 8.
Mi Botón	Númerico	Este es el nombre de un botón. Devuelve el valor actual del botón: 1 si se hizo clic en el botón, 0 si no.
!25/1/97! o !1997/01/25!	Fecha	Esta es una constante de fecha para la fecha 25/1/97 (25 de enero, 1997).
Current date + 30	Fecha	Esta es una expresión fecha que utiliza la función **Current date** para recuperar la fecha actual. el comando "Current date" para obtener la fecha actual. Añade 30 días a la fecha actual y devuelve la nueva fecha.
?8:05:30?	Hora	Esta es una constante de hora que representa 8 horas, 5 minutos, y 30 segundos.
?2:03:04? + ?1:02:03?	Hora	Esta expresión añade una hora a otra y devuelve la hora 3:05:07.
True	Booleano	Este comando devuelve el valor booleano TRUE.
10 # 20	Booleano	Esta es una comparación lógica entre dos números. El signo número (#) significa "es diferente de". Como 10 "no es igual a" 20, la expresión devuelve TRUE.
"ABC" = "XYZ"	Booleano	Esta es un comparación lógica entre dos cadenas. No son iguales, entonces la expresión devuelve FALSE.
Mi Imagen + 50	Imagen	Esta expresión toma la imagen en Mi Imagen, la mueve 50 píxeles a l derecha, y devuelve la imagen resultante.
->[Personas]Nombre	Puntero	Esta expresión devuelva un puntero al campo llamado [Personas]Nombre.
Tabla (1)	Puntero	Este comando retorna un puntero a la primera tabla.
JSON Parse(MiCadena)	Objeto	Este comando devuelve MiCadena en forma de objeto (si el formato es adecuado)
[#cmd id="1218"/] (MyJSONArray)	Colección	Este es un comando que devuelve MyJSONArray como una colección (si el formato es adecuado)

Tipos de datos

Los campos, variables, y expresiones de 4D pueden ser de los siguientes tipos de datos:

Tipos de datos	Campo	Variable	Expresión
Cadena (ver nota 1)	Sí	Sí	Sí
Numérico (ver nota 2)	Sí	Sí	Sí
Fecha	Sí	Sí	Sí
Hora	Sí	Sí	Sí
Booleano	Sí	Sí	Sí
Imagen	Sí	Sí	Sí
Puntero	No	Sí	Sí
BLOB (ver nota 3)	Sí	Sí	No
Array (ver nota 4)	No	Sí	No
Entero 64 bits (ver nota 5)	Sí	No	No
Flotante (ver nota 5)	Sí	No	No
Objeto	Sí	Sí	Sí
Colección	No	Sí	Sí
Indefinido	No	Sí	Sí
Null	No	No	Sí

Notas:

1. Una cadena puede ser un campo alfanumérico, una variable de longitud fija, o un campo o variable tipo Texto.
2. Un numérico puede ser una variable o campo de tipo Real, Entero, y Entero largo.
3. BLOB es la abreviación de Binary Large OBject. Para mayor información sobre BLOBs, consulte la sección **Comandos BLOB**.
4. Los arrays pueden ser de todo tipo. Para mayor información, consulte la sección **Arrays**
5. Los tipos Entero 64 bits y Flotante, sólo son manejados vía SQL. No es recomendable trabajar con ellos vía el lenguaje 4D, porque en este caso ellos se convierten en tipo Real lo cual puede producir pérdida de precisión.

Cadena

Cadena es término genérico que se utiliza para:

- las variables o campos de tipo alfanumérico: un campo alfanumérico puede contener de 0 a 255 caracteres (el límite se fija durante la definición del campo).
- las variables o campos tipo Texto: un campo, una variable o una expresión de tipo texto puede contener de 0 a 2 GB de texto.
- toda expresión de tipo Alfa o Texto

No hay diferencia entre una variable alfanumérica y una variable texto.
Puede asignar una variable alfa a un texto y viceversa; 4D efectúa la conversión, truncando los valores si es necesario. Puede mezclar cadena y texto en las expresiones.

Nota: en el manual del Lenguaje 4D, los parámetros de tipo Alfa y Texto en las descripciones de comandos son llamados Cadena, excepto cuando se especifique de otra forma.

Numérico

Númerico es un término genérico que se utiliza para:

- campos, variables o expresiones de tipo Real
- campos, variables o expresiones de tipo Entero
- campos, variables o expresiones de tipo Entero largo

El rango de números de tipo Real es ±*1.7e*±*308* (13 cifras significativas)
El rango de números de tipo Entero (2-bytes) es *-32,768..32,767* (*2^15..(2^15)-1*)
El rango de números de tipo Entero largo (4-bytes) es *-2^31..(2^31)-1*

Puede asignar todo número de tipo numérico a otro número de otro tipo numérico; 4D hace la conversión, truncando o redondeando si es necesario. Sin embargo, cuando los valores se salen del rango, la conversión no devolverá un valor válido. Puede mezclar tipos de datos numéricos en expresiones.

Nota: en el Manual del lenguaje 4D, sin importar el tipo de dato, los parámetros de tipo Real, Entero, y Entero largo en las descripciones de los comandos son llamados numéricos, excepto cuando se establezca de otra manera.

Fecha

- Un campo, variable o expresión tipo Fecha puede estar en el rango de 1/1/100 a 12/31/32,767.
- Utilizando la versión en español de 4D, una fecha se estructura día/mes/año.
- Si un año tiene sólo dos dígitos, se asume que el siglo es el 20 si el valor es mayor o igual a 30, y el 21 si el valor es menor de 30 (este comportamiento por defecto se puede cambiar utilizando el comando **SET DEFAULT CENTURY**).
- Aunque el modo de representación de fechas por **C_DATE** permite trabajar con fechas hasta el año 32.767, ciertas operaciones que pasan por el sistema imponen un límite inferior.

Nota: en el Manual del lenguaje de 4D, los parámetros de tipo Fecha en las descripciones de los comandos son llamados Fecha, excepto cuando se establezca de otra forma.

Conversión de fechas JavaScript

Las fechas en JavaScript son objetos, son enviadas a 4D como texto que contiene su forma JSON como cualquier otro objeto. Este principio se aplica, en particular, cuando se utilizan las funcionalidades **4D Mobile** o **Área web**.
La forma JSON del objeto Date JavaScript sigue la norma ISO 8601, por ejemplo, 2013-08-23T00:00:00Z".

Es su responsabilidad convertir este texto en una fecha 4D (C_DATE). Están disponibles dos soluciones:

- Utilizar el comando **JSON Parse**:

```
C_TEXT($1) // recepción de una fecha en formato ISO
C_DATE($d)
$d:=JSON Parse("\""+$1+"\"";ls date))
```

- Utilizar el comando **Date**:

```
C_TEXT($1) // recepción de una fecha en formato ISO
C_DATE($d)
$d:=Date($1)
```

Observe la diferencia entre estas dos soluciones: **JSON Parse** respeta el modo de conversión definido por el comando **SET DATABASE PARAMETER** (si lo hay), mientras que **Date** no está sujeta a este. La conversión con el comando **Date** siempre tiene en cuenta la zona horaria local.

Nota: a partir de 4D v16 R6, si la configuración de almacenamiento de fecha actual es "tipo fecha", las cadenas fecha JSON en formato "AAAA-MM-DD" se manejan automáticamente como valores de fecha por los comandos **JSON Parse** y **Date**. Para más información sobre esta configuración, consulte la opción "Utilizar tipo de fecha en lugar de formato de fecha ISO en objetos" en **Página Compatibilidad**.

Hora

- Un campo, variable, o expresión tipo Hora puede estar en el rango de 00:00:00 a 596 000:00:00.
- Utilizando la versión en español de 4D, una hora se estructura bajo la forma hora:minutos:segundos.
- Las horas están en formato de 24 horas.
- Un valor de tipo Hora puede tratarse como un número. El número que devuelve es el número de segundos que el tiempo representa. Para mayor información, consulte la sección .

Nota: en el Manual del lenguaje de 4D, los parámetros de tipo Hora en las descripciones de los comandos son llamados Hora, excepto cuando se establezca de otra forma.

Booleano

Un campo, variable o expresión booleano puede ser VERDADERO o FALSO.

Nota: en el Manual de lenguaje, los parámetros de tipo Booleano en las descripciones se llaman Booleanos, a menos de que establezca de otra forma.

Imagen

Un campo, variable o expresión de tipo Imagen puede contener imágenes Windows o Macintosh. En general, esto incluye cualquier imagen que se pueda colocar en el Portapapeles o leer desde el disco utilizando comandos de 4D o de un plug-in.

Nota: en el Manual del lenguaje 4D, los parámetros de tipo imagen en las descripciones de los comandos son llamados Imagen, a menos de que se establezca de otra forma.

Puntero

Una variable o expresión de tipo puntero es una referencia a otras variables (incluyendo arrays y elementos de array), tablas o campos. No hay campos de tipo Puntero.

Para mayor información sobre Punteros, consulte la sección **Punteros**.

Nota: en el Manual de lenguaje, los parámetros de tipo Puntero en las descripciones de comandos son llamados Puntero excepto cuando se establezca de otra forma.

BLOB

Un campo o variable de tipo BLOB es una serie de bytes (de un lago de 0 a 2 GB) que puede direccionar individualmente o utilizando los . No hay expresiones de tipo BLOB.

Nota: en el Manual de lenguaje de 4D, los parámetros BLOB en las descripciones de los comandos se llaman BLOB.

Objeto

Las variables, campos o expresiones de tipo Objeto pueden contener varios tipos de datos. La estructura de los objetos 4D "nativos" se basa en el principio clásico de los pares "propiedad/valor" (también llamado "atributo/valor). La sintaxis de estos

objetos se basa en JSON, pero no la sigue completamente.

- Un **nombre** de propiedad es siempre texto, por ejemplo "Nombre" (hasta 255 caracteres, sensible a las mayúsculas y minúsculas).
- Un **valor** de propiedad puede ser del siguiente tipo:
  - número (Real, Entero, etc)
  - texto
  - array (texto, real, booleano, objeto, puntero)
  - nulo
  - Booleano
  - puntero (almacenado como tal, evaluado utilizando el comando **JSON Stringify** o al copiar),
  - fecha (tipo fecha o cadena de formato de fecha ISO): consulte **Página Compatibilidad**, "Utilizar el tipo fecha en lugar del formato de fecha ISO en objetos".
  - objeto (los objetos pueden estar anidados en varios niveles)
  - imagen(*)
  - colección

Atención: recuerde que los nombres de atributos diferencian entre mayúsculas y minúsculas.

Para manejar variables, campos o expresiones de tipo Objeto puede utilizar la notación objeto (ver **Uso de la notación objeto**) o los comandos **Objetos (Lenguaje)** de 4D, tales como **OB Get** y **OB SET**. Tenga en cuenta que los comandos específicos del tema **Búsquedas** tales como **QUERY BY ATTRIBUTE**, **ORDER BY ATTRIBUTE** y **QUERY SELECTION BY ATTRIBUTE** se puede utilizar para realizar el procesamiento en campos objeto.

Dado que los campos Objeto normalmente se basan en texto, los contenidos de un campo Objeto se muestran en un formulario 4D de forma predeterminada como texto y se formatean en JSON.

(*) Cuando se expone como texto en el depurador o se exporta a JSON, las propiedades del objeto imagen se muestran como "[object Picture]". Preste atención al hecho de que al guardar el registro se guardará la cadena "[object Picture]" en el atributo.

Nota: para trabajar con objetos JSON, puede utilizar los comandos encontrados en el tema "**JSON**".

Colección

Una variable Colección puede contener una lista ordenada de valores de varios tipos, por ejemplo:

```
C_COLLECTION($col)
$col:=New collection("Ford";"Renault";"Nissan";500;100;True)
 //$col=["Ford","Renault","Nissan",500,100,true]
```

Los tipos de valores soportados son texto, número, objeto, array, booleano, colección o nulo. No hay ninguna expresión o campo de tipo Colección.

Para administrar las variables del tipo Colección, necesita utilizar la notación de objeto (ver **Uso de la notación objeto**) y los comandos del tema **Colecciones**.

- Usted accede a los elementos de la colección a través de su número de elemento (índice),
- Para designar un elemento, use la siguiente sintaxis: myCollection[*N*], donde *N* es el índice del elemento de colección
- **Atención:** el índice del elemento de colección comienza en 0

Ejemplo:

```
C_COLLECTION($col)
$col:=New collection("Ford";"Renault";"Nissan")
$col[1]:="BMW"
 //$col=["Ford","BMW","Nissan"]
```

Las variables Colección almacenan arrays JSON. Un array JSON es una colección de valores separados por comas de cualquier tipo, por ejemplo:

```
C_COLLECTION($c1;$c2)
C_TEXT($json1;$json2)
$c1:=JSON Parse("[\"Ford\",\"Renault\",\"Nissan\",500,100,true]")
$json1:=JSON Stringify($c1)
 //$json1=["Ford","Renault","Nissan",500,100,true]
$c2:=JSON Parse("[1,2,3,\"a\",\"b\",\"c\"]")
$json2:=JSON Stringify($c2)
 //$json2=[1,2,3,"a","b","c"]
```

Indefinido

Indefinido no es realmente un tipo de datos. Denota una variable que no ha sido definida aún. Una función (un método de proyecto que devuelve un resultado) puede devolver un valor indefinido, dentro del método, el resultado de la función (*$0*) se asigna a una expresión indefinida (una expresión calculada con por lo menos una variable indefinida). Un campo no puede ser indefinido. (El comando **Undefined** siempre devuelve False a un campo).

Null

Null es un tipo de datos especial con un solo valor posible: **null**. Este valor es devuelto por una expresión que no contiene ningún valor.

Desde el punto de vista de la base de datos 4D, un valor **null** expresa el hecho de que el valor del dato es desconocido. No significa que el valor está en blanco, o vacío ("" para una cadena, o 0 para un entero largo son valores en blanco). En la base de datos 4D, los valores nulos en campos (excepto los atributos de los campos Objeto) son manejados únicamente por el motor SQL. Una opción de campo específica le permite configurar cómo la base de datos debe manejar este valor (**Mapear valores NULOS a valores vacíos**) y puede definir o leer los valores nulos usando los comandos **Is field value Null** y **SET FIELD VALUE NULL**.

En el lenguaje 4D y para los atributos de los campos objeto, los valores **null** se gestionan mediante la función **Null**. Esta función se puede utilizar con las siguientes expresiones para definir o comparar el valor nulo:

- Atributos objeto
- Elementos de la colección
- Variables de tipo objeto, colección, puntero o imagen

Array

Un array no es realmente un tipo de datos. Los diferentes tipos de arrays (como Array entero, Array texto, etc.) se agrupan bajo este título. Los arrays son variables, no hay campos ni expresiones de tipo Array. Para mayor información sobre arrays, consulte la sección **Arrays**.

Nota: en el Manual de lenguaje 4D, los parámetros de tipo Array en las descripciones de los comandos son llamados Arrays, excepto cuando se establezca de otra forma (por ejemplo Array Texto, Array numérico, ...).

Convertir los tipos de datos

El lenguaje de 4D contiene operadores y comandos para convertir tipos de datos en otros tipos, en la medida en que las conversiones tengan sentido. El lenguaje 4D se asegura de la verificación de los tipos de datos. Por ejemplo, no puede escribir: *"abc"+0.5+!12/25/96!-?00:30:45?*. Esto generará errores de sintaxis.

La siguiente tabla lista los tipos de datos básicos, los tipos de datos en los que se pueden convertir y los comandos a utilizar para hacerlo:

Tipos a convertir	Convertir en cadena	Convertir en número	Convertir en fecha	Convertir en Hora
Cadena (*)		**Num**	**Date**	**Time**
Numérico (**)	**String**			
Fecha	**String**			
Hora	**String**			
Booleano		**Num**		
Objeto	**JSON Stringify**			
Colección	**JSON Stringify**			

(*) Las cadenas formateadas en JSON pueden convertirse en datos escalares, objetos, o colecciones, utilizando el comando **JSON Parse**.
(**) Los valores de tipo Hora pueden tratarse como números.

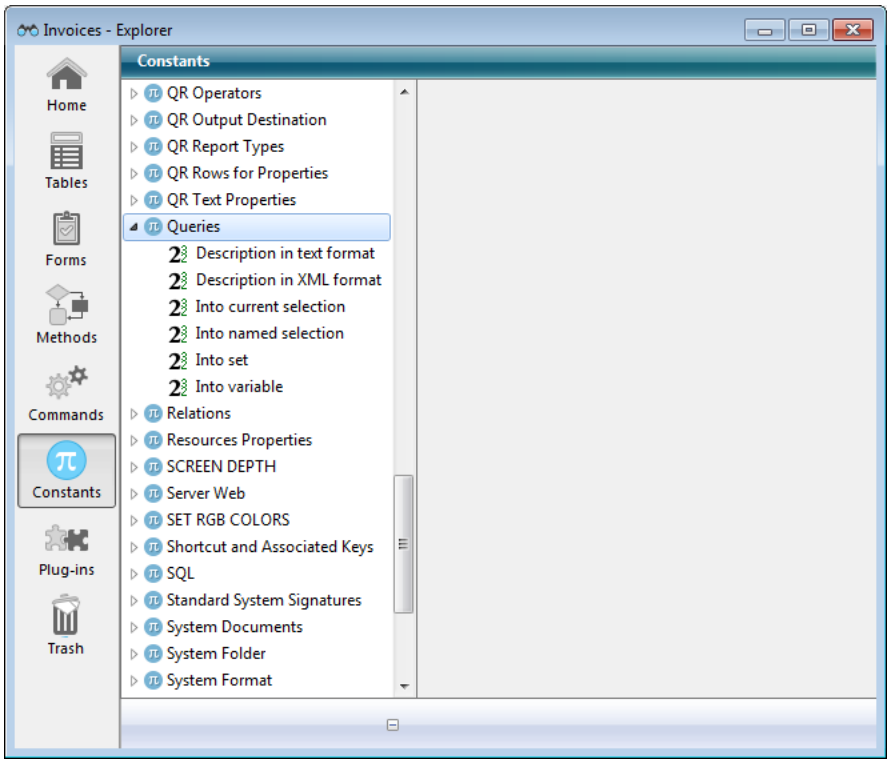
Nota: además de las conversiones de datos listadas en esta tabla, se pueden obtener conversiones de datos más sofisticadas combinando operadores y otros comandos.

Constantes

Una constante es una expresión que tiene un valor fijo. Hay dos tipos de constantes: **constantes predefinidas** que selecciona por nombre y **constantes literales** para las cuales debe introducir un valor.

Constantes predefinidas

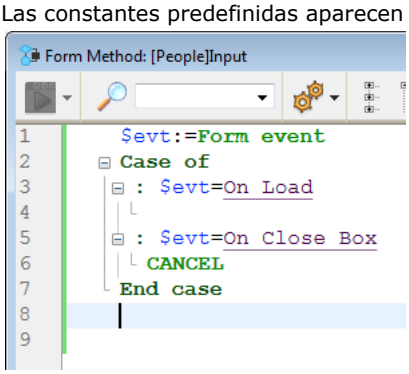
4D ofrece un conjunto de **constantes predefinidas.** Estas constantes están listadas en la ventana del Explorador:



Para utilizar una constante predefinida en una ventana del **editor de métodos**:

- Arrastre y suelte la constante desde la ventana del Explorador a la ventana del editor de métodos.
- Introduzca directamente su nombre en la ventana del editor de métodos. La función de entrada predictiva sugiere las constantes que corresponden al contexto de programación.

Las constantes predefinidas aparecen subrayadas por defecto en el editor de métodos y en la ventana del depurador:



En la ventana anterior, On Load, por ejemplo, es una constante predefinida.

Constantes literales

Las constantes literales pueden ser de cuatro tipos de datos:

- Cadena
- Numérico
- Fecha
- Hora

Constantes cadena

Una constante de tipo cadena está entre comillas ("..."). Estos son algunos ejemplos de constantes cadena:

"Añadir registros"
"Nose encontraron registroa."
"Factura"

Una cadena vacía se especifica por la sucesión de comillas sin nada entre ellas ("").

Constantes numéricas

Una constante numérica se escribe como un número real. Estos son algunos ejemplos de constantes numéricas:

27
123.76
0.0076

Los números negativos se especifican con el signo (−). Por ejemplo:

−27
−123.76
−0.0076

Nota: desde 4D v15, el separador decimal por defecto es un punto (.), Sin importar el idioma del sistema. Si ha marcado la opción "Usar configuración del sistema regional" (ver **Página Métodos**), debe utilizar el separador definido en su sistema.

Constantes Fecha

Una constante fecha se encuentra entre signos de admiración (!…!). Desde 4D v15, una fecha debe estar estructurada utilizando el formato ISO (!YYYY-MM-DD!). Estos son algunos ejemplos de constantes fecha:

!1976-01-01!
!2004-09-29!
!2015-12-31!

Una fecha nula se escribe *!00-00-00!*

Consejo: el editor de métodos incluye un atajo para introducir una fecha nula. Para digitar una fecha nula, digite el carácter signo de admiración (!) y presione Intro.

Notas:

- Por razones de compatibilidad, 4D acepta que el año sea de dos dígitos. Un año de dos dígitos se asume que pertenece al siglo 20 o al 21 basados en si es mayor o menor que 30, a menos que este número por defecto haya sido cambiado utilizando el comando **SET DEFAULT CENTURY**.
- Si ha marcado la opción "Usar configuración del sistema regional" (ver **Página Métodos**), debe utilizar el formato de fecha definido en su sistema. Por lo general, en un ambiente de Estados Unidos, las fechas se introducen mes/día/año, con una barra "/" que separa los valores.

Constantes hora

Una constante hora se encuentra entre signos de interrogación (?...?).

En la versión en castellano de 4D, una constante hora se ordena hora:minuto:segundo, separando los valores con dos puntos (:). Las horas se guardan en formato de 24 horas.

Estos son algunos ejemplos de constantes horas:

?00:00:00? ` media noche
?09:30:00? ` 9:30 de la mañana
?13:01:59? ` 13 horas, 1 minuto, y 59 segundos

Una hora nula se escribe ?00:00:00?

Consejo: el editor de métodos incluye un atajo para introducir una hora nula. Para digitar una hora nula, introduzca el carácter interrogante (?) y presione Intro.

Variables

Los datos en 4D pueden almacenarse de dos maneras diferentes. Los campos almacenan los datos en el disco, de manera permanente; las variables almacenan los datos en memoria, de manera temporal.

Cuando define su base 4D, usted especifica los nombres y tipos de campos que quiere utilizar. Es prácticamente lo mismo para las variables, usted también les da diferentes nombres y tipos.

Los siguientes tipos de variables corresponden a cada uno de los tipos de datos:

- Alfa(*) o texto: cadena alfanumérica de hasta 2GB de texto
- Entero: número entero entre -32768 y 32767
- Entero largo: número entero entre -2^31 y (2^31)-1
- Real: número real entre ±1.7y±308 (13 cifras significativas)
- Fecha: fecha entre 1/1/100 y 31/12/32767
- Time: hora entre 00:00:00 y 596000:00:00 (segundos desde la media noche)
- Booleano: Verdadero o Falso
- Imagen: toda imagen Windows o Macintosh
- Objeto: conjunto de pares "propiedad/valor" estructurados en un formato de tipo JSON
- Colección: lista ordenada de valores de diferentes tipos
- BLOB (Objeto Binario Largo): Series de bytes hasta de 2 GB de tamaño
- Puntero: un puntero hacia una tabla, campo, variable, Array, o elemento de Array

(*) En modo Unicode, los tipos de variable Alfa y Texto son idénticos. En modo no Unicode (modo compatibilidad), un Alfa es una cadena alfanumérica fija de hasta 255 caracteres.

Puede visualizar las variables (excepto punteros y BLOB) en la pantalla, introducir datos en ellas e imprimirlas en informes. De estas formas, las variables editables y no editables se comportan como campos, y los mismos controles integrados están disponibles cuando las crea:

- Formatos de salida
- Validación de datos, tales como filtros de entrada y valores por defecto
- Filtros de caracteres
- Listas de selección (listas jerárquicas)
- Valores editables y no editables

Las variables también pueden servir para:

- controlar botones (botones, casillas de selección, botones radio, botones 3D, etc.)
- controlar termómetros, reglas y dials
- controlar los list box, las áreas de desplazamiento, los menús pop-up y las listas desplegables
- controlar listas y menús jerárquicos
- controlar las rejillas de botones, pestañas, botones de imagen, etc.
- visualizar los resultados de cálculos que no necesitan guardarse.

Crear variables

Puede crear variables simplemente utilizándolas; no necesita definirlas formalmente como lo hace con los campos. Por ejemplo, si quiere crear una variable que contenga la fecha actual más 30 días, sólo tiene que escribir:

```
MiFecha:=Current date+30
```

4D crea **MiFecha** y contiene la fecha que usted necesita. La línea de código se lee "MiFecha es igual a la fecha actual más 30 días." Puede utilizar **MiFecha** cada vez que la necesite en su base. Por ejemplo, podría necesitar guardar los datos de la variable en un campo del mismo tipo:

```
[MiTabla]MiCampo:=MiFecha
```

Sin embargo, generalmente es recomendable definir explícitamente el tipo de una variable. Para mayor información sobre la declaración de variables en una base, consulte la sección .

Asignar valores a las variables

Puede dar valores a las variables y recuperarlos. Dar un valor a una variable se llama **asignar un valor a una variable** y se hace con el *operador de asignación* (*:=*). El operador de asignación también se utiliza para asignar valores a los campos.

El operador de asignación es el primer medio para crear una variable y darle un valor. Usted escribe el nombre de la variable que quiere crear al lado izquierdo del operador de asignación. Por ejemplo:

```
MiNúmero:=3
```

crea la variable **MiNúmero** y coloca el número *3* en ella. Si **MiNúmero** ya existe, sólo se coloca el número 3 en la variable.

Desde luego, las variables no serían muy útiles si no pudiera recuperar los valores que contienen. Una vez más, utilice el operador de asignación. Si necesita colocar el valor de **MiNúmero** en un campo llamado *[Productos]Tamaño*, basta con escribir **MiNúmero** a la derecha del operador de asignación:

```
[Productos]Tamaño:=MiNúmero
```

En este caso, *[Productos]Tamaño* sería igual a 3. Este ejemplo es bastante sencillo, pero ilustra la forma fundamental en que los datos se transfieren de un lugar a otro utilizando el lenguaje.

Importante: no confunda el operador de asignación (*:=*) con el operador de comparación, igual (=). La asignación y la comparación son operaciones diferentes. Para mayor información sobre operadores de comparación, vea la sección **Operadores**.

Variables locales, proceso, e interproceso

Puede crear tres tipos de variables: variables **locales**, variables **proceso**, y variables **interproceso**. La diferencia entre los tres tipos de variables es su alcance, o los objetos para los cuales están disponibles.

Variables locales

Una variables local, como su nombre lo indica, es local a un método, accesible sólo dentro del método en el cual fue creada e inaccesible fuera de ese método. Una variable local a un método era llamada anteriormente "local en alcance." Las variables locales se utilizan para restringir una variable de manera que funciona sólo dentro del método.

Podría querer utilizar una variable local para:

- Evitar conflictos de nombres con otras variables
- Utilizar temporalmente los valores
- Reducir el número de variables proceso

El nombre de una variable local siempre comienza con el signo dólar ($) y puede contener hasta 31 caracteres adicionales. Si introduce un nombre más largo, 4D lo trunca a la longitud apropiada.

Cuando trabaja en una base con muchos métodos y variables, con frecuencia se encuentra con que necesita utilizar una variable sólo dentro del método en el que está trabajando. Puede crear y utilizar una variable local en el método sin preocuparse por la existencia de otra variable con el mismo nombre en otro método.

Con frecuencia, en una base de datos, el usuario necesita información puntual. El comando **Request** puede obtener esta información. Muestra una caja de diálogo con un mensaje que le pide al usuario responder. Cuando el usuario introduce la respuesta, el comando devuelve la información que el usuario introdujo. Generalmente no es necesario conservar esta información en sus métodos por mucho tiempo. Esta es una forma típica de utilizar una variable local. Este es un ejemplo:

```
$vsID:=Request("Por favor introduzca su número de identificación ID:")
If(OK=1)
   QUERY([Personas];[Personas]ID=$vsID)
End if
```

Este método simplemente le pide al usuario introducir un número de identificación. La respuesta está ubicada en una variable local, *$vsID*, y luego busca el número de identificación que el usuario introdujo. Cuando el método termina, la variable local *$vsID* se borra de la memoria. Este funcionamiento está bien, porque la variable sólo se necesita una vez y sólo en este método.

Variables proceso

Una variable proceso está disponible sólo dentro de un proceso. Es accesible en el método de proceso y en cualquier otro método llamado dentro del proceso.

Una variable proceso no tiene un prefijo antes de su nombre y puede tener hasta 31 caracteres.

En modo interpretado, las variables se mantienen dinámicamente, son creadas y borradas de la memoria "rápidamente." En modo compilado, todos los procesos que usted crea (procesos usuario) comparten la misma definición de variables proceso, pero cada proceso tiene su propia instancia para cada variable. Por ejemplo, la variable *miVar* es una variable en el proceso **P_1** y otra en el proceso **P_2**.

Un proceso puede leer y escribir variables proceso de otros procesos utilizando los comandos **GET PROCESS VARIABLE** y **SET PROCESS VARIABLE**. Recomendamos restringir el uso de estos comandos a las situaciones para los cuales fueron creados en 4D:

- Comunicación interprocesos en lugares específicos en su código
- Gestión de interprocesos arrastrar y soltar
- En cliente/servidor, comunicación entre los procesos en equipos cliente y los procesos guardados en ejecutados en el servidor.

Para mayor información, consulte la sección **Procesos** y la descripción de estos comandos.

Variables interproceso

Las variables interproceso están disponibles en toda la base y durante todos los procesos. Principalmente se utilizan para compartir información entre procesos.

El nombre de una variable interproceso siempre comienza con el símbolo (<>), un signo "menor que" seguido por un signo "mayor que", seguido por hasta 31 caracteres.

Nota: esta sintaxis puede utilizarse en Windows y Macintosh. Además, en Macintosh únicamente, puede utilizar el símbolo diamante (Opción-Mayús-V en teclado en castellano).

En cliente/servidor, cada equipo (equipos cliente y servidor) comparten la misma definición de variables interproceso, pero cada equipo tiene una instancia diferente para cada variable.

Variables objeto en los formularios

En el editor de formularios, cada objeto activo - botón, botón radio, casilla de selección, área de desplazamiento, termómetro, etc. - está identificado por un nombre de objeto y se asocia automáticamente a una variable (o una expresión). Por defecto, la variable no define la creación del objeto: será creada dinámicamente al cargar el formulario (ver abajo). Puede, si lo desea,

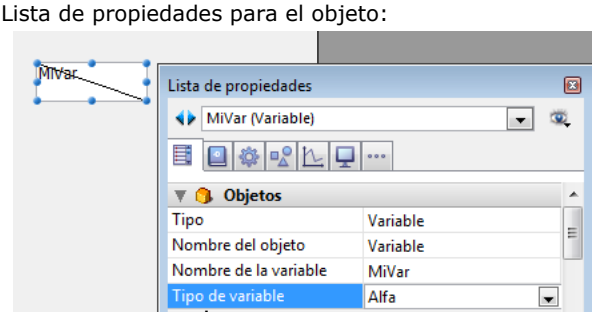
nombrar la variable en la lista de propiedades con el fin de crearla. Por ejemplo, si crea un botón llamado *MiBoton*, también se crea una variable llamada *MiVarBoton* (también puede utilizar el mismo nombre que el objeto).

Las variables de objeto de formulario le permiten controlar y monitorear los objetos. Por ejemplo, cuando se hace clic en un botón, su variable toma el valor 1; de lo contrario está en 0. La variable asociada a un termómetro o dial le permite leer y cambiar los valores actuales. Por ejemplo, si arrastra un termómetro a un nuevo valor, el valor de la variable cambia para reflejar el nuevo valor. De la misma forma, si un método cambia el valor de la variable, el termómetro se dibuja nuevamente para mostrar el nuevo valor.

Para mayor información sobre variables y formularios, consulte el Manual de Diseño 4D y la sección **Eventos de formulario**.

Variables dinámicas

Puede dejar a 4D crear dinámicamente variables asociadas a sus objetos de formulario (botones, variables editables, casillas de selección, etc.) y de acuerdo a sus necesidades. Para hacerlo, simplemente deje vacío el campo "Nombre de la variable" en la Lista de propiedades para el objeto:



Cuando no se le da nombre a una variable, al cargar el formulario, 4D crea una nueva variable para el objeto, con un nombre calculado que es único en el espacio de las variables proceso del intérprete (lo cual significa que este mecanismo puede utilizarse incluso en modo compilado). Esta variable temporal se destruirá cuando se cierre el formulario.

Para que este principio funcione en modo compilado, es imperativo que las variables dinámicas se definan explícitamente. Hay dos formas de hacer esto:

- definir el tipo vía el menú "Tipo de variable" de la lista de propiedades.
  Nota: cuando se le da nombre a la variable, el menú "Tipo de variable" no define en realidad la variable pero permite la actualización de las opciones de la lista de propiedades (excepto para las variables imagen). Para definir una variable con nombre, es necesario utilizar los comandos del tema **Compilador**.
- puede utilizar un código de utilización específico durante la carga del formulario, utilizando por ejemplo el comando **VARIABLE TO VARIABLE**:

```
If(Form event=On Load)
   C_TEXT($init)
   $Ptr_object:=OBJECT Get pointer(Object named;"Comentarios")
   $init:=""
   VARIABLE TO VARIABLE(Current process;$Ptr_object->;$init)
End if
```

Nota: si define una variable dinámica, seleccione el valor **Ninguno** en el menú "Tipo de variable" y no utilice código de inicialización, un error de digitación será devuelto por el compilador.

En el código 4D, las variables dinámicas son accesibles vía un puntero obtenido con el comando **OBJECT Get pointer**. Por ejemplo:

```
// asignar la hora 12:00:00 a la variable del objeto "tstart"
$p :=OBJECT Get pointer(Object named;"tstart")
$p->:=?12:00:00?
```

Hay dos ventajas con este mecanismo:

- Permite el desarrollo de componentes de tipo "subformulario" que pueden ser utilizados varias veces en un mismo formulario local. Tomemos como ejemplo el caso de un subformulario *datepicker* insertado dos veces en un formulario local para definir una fecha de inicio y una fecha final. Este subformulario utilizará objetos para la selección de la fecha del mes y el año. Será necesario para que estos objetos trabajen con las variables diferentes para la fecha de inicio y la fecha final. Dejar que 4D cree su variable con un nombre único es una forma de resolver esta dificultad.
- Permite limitar el uso de la memoria. De hecho, los objetos de formulario sólo trabajan con variables proceso o inter proceso. O en modo compilado, una instancia de cada variable proceso se crea en todos los procesos, incluyendo los procesos del servidor. Esta instancia consume memoria, incluso si el formulario no está siendo utilizado durante la sesión. Por lo tanto, permitir que 4D cree dinámicamente las variables al cargar los formularios permite economizar memoria.

Nota: cuando no hay nombre de variable, el nombre del objeto se muestra entre comillas en el editor de formularios (cuando el objeto muestra por defecto un nombre de variable).

Variables de sistema

4D mantiene un número de variables llamadas **variables de sistema**. Estas variables le permiten controlar muchas operaciones. Todas las variables sistema son variables de proceso, disponibles al interior de un solo proceso.

La variable de sistema mas importante es la variable sistema **OK**. Como su nombre lo indica, esta variable le dice si todo está bien en un proceso en particular. ¿Se guardó el registro? ¿Se completó la operación de importación? ¿El usuario hizo clic en el botón OK? La variable sistema OK toma el valor 1 cuando una tarea se completa correctamente, y 0 cuando ocurre lo contrario.

Para mayor información sobre variables sistema, consulte la sección **Variables de sistema**.

Variables sistema

4D administra **variables de sistema**, las cuales le permiten controlar la ejecución de diferentes operaciones. Todas las variables sistema son variables proceso que sólo son accesibles dentro de un proceso. Esta sección describe las variables sistema de 4D.
Para mayor información sobre este tipo de variables, consulte el párrafo **Variables sistema** en la sección **Guía de declaración**.

OK

La variable de sistema OK es la más comúnmente utilizada. Generalmente, toma el valor 1 cuando una operación se ejecuta con éxito. Toma el valor 0 cuando la operación falla. La mayoría de los comandos 4D modifican el valor de la variable sistema **OK**. Consulte la descripción de cada comando para saber si afecta esta variable sistema.

En esta documentación, el pictograma  indica que un comando modifica el valor de la variable OK. Puede hacer clic en esta imagen para generar una lista de todos los comandos concernientes.

Document

Document contiene el "nombre completo" (ruta de acceso+nombre) del último archivo abierto o creado utilizando los siguientes comandos:

Append document	**BUILD APPLICATION**
Create document	**Create resource file**
EXPORT DATA	**EXPORT DIF**
EXPORT SYLK	**EXPORT TEXT**
IMPORT DATA	**IMPORT DIF**
IMPORT SYLK	**IMPORT TEXT**
GET DOCUMENT ICON	**LOAD SET**
LOAD VARIABLES	**Open document**
Open resource file	**PRINT LABEL**
QR REPORT	**READ PICTURE FILE**
SAVE VARIABLES	**SAVE SET**
Select document	**SELECT LOG FILE**
SET CHANNEL	**USE CHARACTER SET**
WRITE PICTURE FILE	

FldDelimit

FldDelimit contiene el código ASCII que se utilizará como un separador de campos cuando se importa o exporta texto. Por defecto, este valor es 9, es decir el código ASCII para la tecla Tab. Para utilizar un separador de campos diferente, asigne un nuevo valor a **FldDelimit**.

RecDelimit

RecDelimit contiene el código ASCII del carácter a utilizar como separador de registros cuando se importa o exporta texto. Por defecto, este valor es 13, el cual es el código ASCII para la tecla Retorno de carro. Para utilizar un separador de registros diferente, asigne un nuevo valor a **RecDelimit**.

Error, Error method, Error line

Estas variables sólo pueden utilizarse en un método de intercepción de errores instalado por el comando **ON ERR CALL**. Si quiere que sean accesibles en el método que provocó el error, copie su valor en sus propias variables proceso.

- **Error**: variable sistema de tipo entero largo. Esta variable contiene el código de error. Los códigos de error de 4D y códigos de errores sistema se listan en la sección **Códigos de error**.
- **Error method**: variable sistema de tipo texto. Esta variable contiene el nombre completo del método que disparó el error.
- **Error line**: variable sistema de tipo entero largo. Esta variable contiene el número de la línea en el origen del error en el método que disparó el error.
- **Error formula**: variable sistema de tipo texto. Esta variable contiene la fórmula de código 4D (texto sin formato), que está en el origen del error. El texto de la fórmula se expresa en el lenguaje actual del código 4D.
  Si no se puede encontrar el código fuente responsable del error, **Error formula** contiene una cadena vacía. Este caso puede ocurrir en las bases de datos compiladas cuando:
  - el código fuente se ha eliminado de la estructura compilada utilizando el generador de aplicaciones.
  - el código fuente está disponible, pero la base de datos fue compilado sin la opción **Control de ejecución**.

MouseDown, MouseX, MouseY, KeyCode, Modifiers y MouseProc

Estas variables sistema sólo pueden utilizarse en un método instalado por el comando **ON EVENT CALL** (excepto **MouseX** y **MouseY** en algunos casos, ver abajo).

- **MouseDown** toma el valor 1 cuando se presiona el botón del ratón. De lo contrario, toma el valor 0.
- ISi el evento es un **MouseDown** (**MouseDown**=1), las variables sistema **MouseX** y **MouseY** se definen para las coordenadas vertical y horizontal respectivamente de la ubicación donde se efectuó el clic. Ambos valores se expresan en píxeles y utilizan el sistema de coordenadas local de la ventana.
  Nota: cuando se hace clic en un campo o variable imagen, las variables sistema **MouseX** y **MouseY** devuelven las coordenadas locales del clic en los eventos de formulario On Clicked, On Double Clicked así como también en los eventos formulario On Mouse Enter y On Mouse Move. Para mayor información consulte la sección **Introducción a las imágenes** y el comando **SVG Find element ID by coordinates**. También cuando se genera el evento On Mouse Up en una imagen, **MouseX** y **MouseY** devuelve las coordenadas locales donde se libera el botón del ratón. Las coordenadas se expresan en píxeles con respecto a la esquina superior izquierda de la imagen (0,0). Para más información, consulte la descripción del comando **Is waiting mouse up**.
- **KeyCode** contiene el código del carácter de la tecla que fue presionada. Si la tecla es una tecla de función, **KeyCode** contiene un código especial. Los códigos de caracteres y los códigos de teclas de función están listados en las secciones **Códigos Unicode**, **EXPORT TEXT** y **Códigos de teclas de función**.
- **Modifiers** contiene los códigos de los modificadores del teclado (**Ctrl/Comando**, **Alt/Opción**, **Mayús, Bloq Mayús**). Esta variable sólo es significativa en un evento de interrupción instalado por el comando **ON EVENT CALL**.
- **MouseProc** contiene el número del proceso en el cual tomó lugar el último evento.

Punteros

Descripción

Los punteros ofrecen una forma avanzada (en programación) para referirse a datos.

Cuando utiliza el lenguaje, accede a diferentes objetos, en particular tablas, campos, variables y arrays, simplemente utilizando sus nombres. Sin embargo, con frecuencia es útil hacer referencia a estos elementos y acceder a ellos sin conocer sus nombres. Esto es lo que los punteros le permiten hacer.

El concepto detrás de los punteros no es desconocido en la vida diaria. Con frecuencia usted se refiere a algo sin saber su identidad exacta. Por ejemplo, podría decirle a un amigo, "Vamos a pasear en tu coche" en lugar de "Vamos a pasear en el coche de matrícula 123ABD." En este caso, está haciendo referencia al coche de matrícula 123ABD utilizando la frase "tu coche." La frase "coche de matrícula 123ABD" es como el nombre de un objeto, y utilizar la frase "tu coche" es como utilizar un puntero para referenciar el objeto.

La capacidad de referirse a algo sin conocer su identidad exacta es muy útil. De hecho, su amigo compra un coche nuevo, la frase "tu coche" seguiría siendo correcta, seguiría siendo un coche y usted podría ir de paseo en él. Los punteros funcionan de la misma manera. Por ejemplo, un puntero podría en un momento referirse un campo numérico llamado **Edad** y más tarde referirse a una variable numérica llamada **Edad antigua**. En ambos casos, el puntero hace referencia a un dato numérico que puede ser utilizado en un cálculo.

Puede utilizar punteros para referenciar tablas, campos, variables, arrays, y elementos de Array. La siguiente tabla la da un ejemplo de cada tipo de dato:

Objeto	Referenciamiento	Utilización	Asignación
Tabla	vpTabla:=->[Tabla]	DEFAULT TABLE(vpTabla->)	n/a
Campo	vpCampo:=->[Tabla]Campo	ALERT(vpCampo->)	vpCampo->:="Juan"
Variable	vpVar:=->Variable	ALERT(vpVar->)	vpVar->:="Juan"
Array	vpArr:=->Array	SORT ARRAY(vpArr->;>)	COPY ARRAY(Arr;vpArr->)
Elemento array	vpElem:=->Array{1}	ALERT (vpElem->)	vpElem->:="Juan"

Utilizar punteros: un ejemplo

Es más fácil explicar el uso de punteros a través de un ejemplo. Este ejemplo muestra cómo acceder a una variable a través de un puntero. Comenzamos creando una variable:

```
MiVar:="Hola"
```

MiVar es ahora una variable que contiene la cadena *"Hola".* Ahora podemos crear un puntero a **MiVar**:

```
MiPuntero:=->MiVar
```

El símbolo -> significa "puntero a." Este símbolo está formado por un guión seguido por un signo "mayor que". En este caso, crea un puntero que referencia o "apunta a" **MiVar**. Este puntero se asigna a **MiPuntero** con el operador de asignación.

MiPuntero es ahora una variable que contiene un puntero a **MiVar**. **MiPuntero** no contiene *"Hola"*, el valor de **MiVar**, pero puede utilizar **MiPuntero** para obtener este valor. La siguiente expresión devuelve el valor en **MiVar**:

```
MiPuntero->
```

En este caso, devuelve la cadena *"Hola"*. El símbolo -> , cuando está ubicado detrás de un puntero, hace referencia al objeto al que se apunta. Esto se llama **desreferenciación**.

Es importante entender que puede utilizar un puntero seguido por el símbolo -> en todas las partes donde pueda utilizar el objeto al que el puntero apunta. Esto significa que puede utilizar la expresiónMiPuntero-> en todas las partes donde podría utilizar la variable original **MiVar**.

Por ejemplo, la siguiente instrucción muestra una caja de diálogo de alerta con la palabra Hola:

```
ALERT(MiPuntero->)
```

Igualmente puede utilizar **MiPuntero** para cambiar los datos en **MiVar**. Por ejemplo, la siguiente instrucción guarda la cadena *"Hasta luego"* en la variable **MiVar**:

```
MiPuntero->:="Hasta luego"
```

Si examina los dos usos de la expresiónMiPuntero->, verá que actúa como si hubiera utilizado **MiVar** en su lugar. En resumen, los siguientes dos tipos de líneas realizan la misma acción—ambos muestran una caja de alerta que contiene el valor actual en la variable **MiVar**:

```
ALERT(MiPuntero->)
ALERT(MVar)
```

Las dos líneas siguientes realizan la misma acción— ambas asignan la cadena *"Hasta luego"* a **MiVar**:

```
MiPuntero->:="Hasta luego"
MiVar:="Hasta luego"
```

Utilizar punteros para referenciar botones

Esta sección describe cómo utilizar un puntero para referenciar un botón. Un botón es (desde el punto de vista del lenguaje) nada más que una variable. Aunque los ejemplos en esta sección utilizan los punteros para referenciar botones, los conceptos que se presentan aplican al uso de todos los tipos de objetos que pueden ser referenciados por un puntero.

Digamos que tiene un número de botones en sus formularios que necesitan ser activados o desactivados. Cada botón tiene una condición asociada que es TRUE o FALSE. La condición indica si hay que desactivar o activar el botón. Puede utilizar una prueba como esta cada vez que necesite activar o desactivar el botón:

```
If(Condición)  ` Si la condición es TRUE···
   OBJECT SET ENABLED(MiBoton)  ` activar el botón
Else  ` De lo contrario···
   OBJECT SET ENABLED(MiBoton)  ` desactivar el botón
End if
```

Puede necesitar utilizar una prueba similar para cada botón que creó, en la cual sólo cambia el nombre del botón. Para ser más eficientes, puede utilizar un puntero para hacer referencia a cada botón y luego utilizar una subrutina para la prueba.

Debe utilizar punteros si utiliza una subrutina, porque no es posible hacer referencia a las variables de los botones de otra manera. Por ejemplo, he aquí un método de proyecto llamado **DEFINIR_BOTON**, el cual hace referencia a un botón con un puntero:

```
  ` Método de proyecto DEFINIR_BOTON
  ` DEFINIR_BOTON ( Puntero ; Booleano )
  ` DEFINIR_BOTON ( -> Botón ; Activado o Desactivado )
  `
  ` $1 – Puntero a un botón
  ` $2 – Booleano. Si TRUE, activa el botón. Si FALSE, desactiva el botón

If($2)  ` Si la condición es TRUE···
   OBJECT SET ENABLED($1->)  ` activar el botón
Else  ` De lo contrario···
   OBJECT SET ENABLED($1->)  ` desactivar el botón
End if
```

Puede llamar al método de proyecto **DEFINIR_BOTON** de esta manera:

```
  ` ...
DEFINIR_BOTON(->bValidar;True)
  ` ...
DEFINIR_BOTON(->bValidar;False)
  ` ...
DEFINIR_BOTON(->bValidar;([Empleados]Apellido#"")
  ` ...
For($vlRadioBoton;1;20)
   $vpRadioBoton:=Get pointer("r"+String($vlRadioBoton))
   DEFINIR_BOTON($vpRadioBoton;False)
End for
```

Utilizar punteros a tablas

En todas las partes donde el lenguaje requiera una tabla, puede utilizar un puntero desreferenciado hacia la tabla.
Para crear un puntero hacia una tabla, escriba una instrucción como esta:

```
TablaPtr:=->[cualquier tabla]
```

Igualmente puede obtener un puntero a una tabla utilizando el comando **Table**. Por ejemplo:

```
TablaPtr:=Table(20)
```

Puede utilizar el puntero desreferenciado en comandos, de esta forma:

```
DEFAULT TABLE(TablaPtr->)
```

Utilizar punteros a los campos

En todas las partes donde el lenguaje requiera campo, puede utilizar un puntero desrefenciado para referenciar el campo. Puede crear un puntero a un campo utilizando una línea como esta:

```
CampoPtr:=->[unaTabla]EsteCampo
```

Igualmente puede obtener un puntero a un campo utilizando el comando **Field**. Por ejemplo:

```
CampoPtr:=Field(1;2)
```

Puede utilizar el puntero desreferenciado en comandos, como este:

```
OBJECT SET FONT(CampoPtr->;"Arial")
```

Utilizar punteros a variables

El ejemplo al comienzo de esta sección ilustra la utilización de un puntero a una variable:

```
MiVar:="Hola"
MiPuntero:=->MiVar
```

Puede utilizar punteros a variables interprocesos, de proceso y, a partir de la versión 2004.1, a variables locales.

Cuando utilice punteros a variables de proceso o a variables locales, debe asegurarse que la variable a la que se apunta esté definida cuando se utilice el puntero. Recuerde que las variables locales se borran cuando el método que las creó completa su ejecución y las variables proceso se borran al final del proceso que las creó. Cuando un puntero llama una variable que no existe, se produce un error de sintaxis en modo interpretado (variable no definida) pero puede generar un error más serio en modo compilado.

Nota sobre variables locales: los punteros a variables locales le permiten ahorrar variables de proceso en muchos casos. Los punteros a variables locales sólo pueden utilizarse dentro del mismo proceso.

En el depurador, cuando visualiza un puntero a una variable local que ha sido declarada en otro método, el nombre del método original está entre paréntesis, después del puntero. Por ejemplo, si escribe en Metodo1:

```
$MiVar:="Buenos días"
Metodo2(->$MiVar)
```

En Metodo2, el depurador mostrará *$1* de esta manera:
$1 ->$MiVar (Metodo1)

El valor de *$1* será:
$MiVar (Metodo1) "Buenos días"

Utilizar punteros a elementos Array

Puede crear un puntero a un elemento Array. Por ejemplo, las líneas siguientes crean un Array y asignan a una variable llamada **ElemPtr un puntero al primer elemento Array :**

```
ARRAY REAL(unArray;10)  ` Crea un array
ElemPtr:=->unArray{1}  ` Crea un puntero al elemento de array
```

Puede utilizar el puntero desreferenciado para asignar un valor al elemento, como este:

```
ElemPtr->:=8
```

Utilizar punteros a arrays

Puede crear un puntero a un Array. Por ejemplo, las líneas siguientes crean un Array y asignan a una variable llamada **ArrPtr un puntero al Array**:

```
ARRAY REAL(unArray;10)  ` Crea un array
ArrPtr:=->unArray  ` Crea un puntero al array
```

Es importante entender que el puntero apunta al Array; no a un elemento del Array. Por ejemplo, puede utilizar el puntero desreferenciado de esta manera:

```
SORT ARRAY(ArrPtr->;>)  ` Ordenar el array
```

Si necesita hacer referencia al cuarto elemento en el Array utilizando el puntero, puede hacer esto:

```
ArrPtr->{4}:=84
```

Utilizar un array de punteros

Con frecuencia es útil tener un Array de punteros que referencian a un grupo de objetos relacionados.

Un ejemplo de utilización de un grupo de objetos es una rejilla de variables en un formulario. Cada variable en la rejilla está numerada secuencialmente, por ejemplo: **Var1**,**Var2**,..., **Var10**. Con frecuencia usted necesita referenciar estas variables indirectamente con un número. Si crea un Array de punteros e inicializa los punteros hacia cada variable, entonces puede referenciar fácilmente las variables. Por ejemplo, para crear un Array e inicializar cada elemento, puede utilizar las siguientes líneas:

```
ARRAY POINTER(apPunteros;10)  ` Crear un array de 10 punteros
For($i;1;10)  ` Bucle una vez por variable
   apPunteros{$i}:=Get pointer("Var"+String($i))  ` Inicializar el elemento array
End for
```

La función **Get pointer** devuelve un puntero al objeto llamado.

Para referenciar una de las variables, utilice los elementos Array. Por ejemplo, para llenar las variables con las siguientes diez fechas (asumiendo que las variables son tipo Fecha), puede utilizar las siguientes líneas:

```
For($i;1;10)  ` Bucle una vez por cada variable
   apPunteros{$i}->:=Current date+$i   ` Asignar las fechas
End for
```

Definir un botón utilizando un puntero

Si tiene un grupo homogéneo de botones radio en un formulario, con frecuencia necesita definirlos rápidamente. No es eficiente referenciar directamente cada uno de ellos por su nombre. Imagine que tiene un grupo de botones radio llamados **Boton1**, **Boton2**,..., **Boton5**.

En un grupo de botones radio, sólo se selecciona un botón. El número de botón seleccionado puede guardarse en un campo numérico. Por ejemplo, si el campo llamado *[Preferencias]Estado* contiene *3*, entonces el **Boton3** se selecciona. En su método de formulario, puede utilizar el siguiente código para definir el botón:

```
Case of
   :(Form event=On Load)
   ` ...
      Case of
         :([Preferencias]Estado=1)
            Boton1:=1
         :([Preferencias]Estado=2)
            Boton2:=1
         :([Preferencias]Estado=3)
            Boton3:=1
         :([Preferencias]Estado=4)
            Boton4:=1
         :([Preferencias]Estado=5)
            Boton5:=1
      End case
   ` ...
End case
```

Un caso particular debe ser probado para cada botón radio. El método puede ser muy largo si tiene muchos botones radio en su formulario. Afortunadamente, puede utilizar punteros para solucionar este problema. Puede utilizar el comando **Get pointer** para devolver un puntero a un botón radio. El siguiente ejemplo utiliza un apuntador de este tipo para referenciar al botón radio seleccionado. Este es el código mejorado:

```
Case of
   :(Form event=On Load)
   ` ...
      $vpRadio:=Get pointer("Boton"+String([Preferencias]Estado))
      $vpRadio->:=1
   ` ...
End case
```

El número del botón radio tratado debe ser almacenado en el campo *[Preferencias]*Estado. Puede hacerlo en el método de formulario para el evento **On Clicked**:

```
[Preferencias]Estado:=Boton1+(Boton2*2)+(Boton3*3)+(Boton4*4)+(Boton5*5)
```

Pasar punteros a métodos

Puede pasar un puntero como un parámetro a un método. Dentro del método, puede modificar el objeto referenciado por el puntero. Por ejemplo, el siguiente método, **TOME DOS**, toma dos parámetros que son punteros. Esto cambia el objeto referenciado por el primer parámetro a caracteres en mayúsculas, y el objeto referenciado por el segundo parámetro a caracteres en minúsculas. Este es el método:

```
` Método de proyecto TOME DOS
` $1 – Puntero a un campo o variable tipo cadena. Pasa la cadena a mayúsculas.
` $2 – Puntero a un campo o variable tipo cadena. Pasa la cadena a minúsculas.
$1->:=Uppercase($1->)
$2->:=Lowercase($2->)
```

La siguiente instrucción utiliza el método **TOME DOS** para cambiar un campo a caracteres en mayúsculas y cambiar una variable a caracteres en minúsculas:

```
TOME DOS(->[Mi Tabla]Mi Campo;->MiVar)
```

Si el campo *[Mi Tabla]Mi Campo* contiene la cadena *"gómez",* cambiaría a la cadena *"GÓMEZ"*. Si la variable **MiVar** contiene la cadena *"HOLA",* cambiaría a la cadena *"hola"*.

En el método **TOME DOS**, y de hecho, cada vez que utiliza punteros, es importante que los tipos de datos de los objetos referenciados sea correcto. En el ejemplo anterior, los punteros deben apuntar a un objeto que contenga una cadena o un texto.

Punteros a punteros

Si en realidad le gusta complicar la cosas, puede utilizar punteros para referenciar otros punteros. Considere este ejemplo:

```
MiVar:="Hola"
PunteroUno:=->MiVar
PunteroDos:=->PunteroUno
(PunteroDos->)->:="Hasta luego"
ALERT((Puntero Dos->)->)
```

Este ejemplo muestra una caja de diálogo de alerta con la palabra *"Hasta luego"*.

Esta es una explicación de cada línea del ejemplo:

- *MiVar:="Hola"*
  --> Esta línea asigna la cadena *"Hola"* en la variable **MiVar**.
- *PunteroUno:=->MiVar*
  --> **PunteroUno** ahora contiene un puntero a **MiVar**.
- *PunteroDos:=->PunteroUno*
  --> **PunteroDos** (una nueva variable) contiene un puntero a **PunteroUno**, el que apunta a **MiVar**.
- *(PunteroDos->)->:="Hasta luego"*
  --> **PunteroDos->** referencia el contenido de **PunteroUno**, el cual referencia a **MiVar**.

Por lo tanto (**PunteroDos->)->** referencia el contenido de **MiVar**. De manera que en este caso, **MiVar** se asigna a *"Hasta luego"*.

- *ALERT ((PunteroDos->)->)*
  --> El mismo procedimiento: **PunteroDos->** referencia el contenido de **PunteroUno**, que a su vez referencia a **MiVar**. Por lo tanto (**PunteroDos->)->** referencia el contenido de **MiVar**. De esta manera, la caja de diálogo de alerta muestra el contenido de **MiVar**.

La siguiente línea coloca *"Hola"* en **MiVar**:

```
(PunteroDos->)->:="Hola"
```

La línea siguiente obtiene *"Hola"* de **MiVar** y la asigna a **NuevaVar**:

```
NuevaVar:=(PunteroDos->)->
```

Importante: el desreferenciamiento múltiple requiere paréntesis.

Convenciones

Esta sección describe las reglas de escritura y de nombres aplicadas a los diferentes identificadores utilizados en el lenguaje 4D (variables, arrays, . El nombre de cada objeto debe respetar las siguientes reglas:

- Un nombre debe comenzar por un carácter alfabético (una letra) o un guión bajo.
- El nombre puede contener caracteres alfabéticos, caracteres numéricos, espacios y guiones bajos.
- Las comas, barras oblicuas, comillas y dos puntos (:) no están permitidos.
- Los puntos(".") y los corchetes ("[ ]") no están permitidos en los nombres de tablas, campos, métodos o variables cuando la notación objeto está activa. (ver **Página Compatibilidad**)
- Los caracteres reservados para utilizar como operadores, tales como el asterisco (*) y el +, no están permitidos.
- 4D ignora los espacios vacíos al final.

Nota: las reglas adicionales deben ser respetadas cuando los objetos deben ser manipulados vía el SQL: sólo se aceptan los caracteres *_0123456789abcdefghijklmnopqrstuvwxyz*, y el nombre no debe incluir las palabras claves SQL (command, attribute, etc.). El área "SQL" del Inspector del editor de estructura indica automáticamente los caracteres no autorizados en el nombre de una tabla o de campo.

Tablas

Designa una tabla ubicando su nombre entre corchetes: *[...]*. El nombre de una tabla puede contener hasta 31 caracteres.

Ejemplos

```
DEFAULT TABLE([Orders])
FORM SET INPUT([Clients];"Entry")
ADD RECORD([Letters])
```

Campos

Designa un campo especificando primero la tabla a la cual pertenece el campo. El nombre del campo se coloca inmediatamente después del nombre de la tabla. El nombre de un campo puede contener hasta 31 caracteres.

Ejemplos

```
[Ordenes]Total:=Sum([Linea]Cantidad)
QUERY([Clientes];[Clientes]Nombre="López")
[Cartas]Texto:=Capitalize text([Cartas]Texto)
```

Variables interproceso

Designa una variable interproceso precediendo el nombre de la variable con los símbolos (<>), un signo "menor que" seguido por un signo "mayor que".

Nota: esta sintaxis se puede utilizar en Windows y Macintosh. Además, en Macintosh únicamente, puede utilizar el carácter diamante (Opción-Mayús-V en teclado en castellano).

Una variable interproceso puede tener hasta 255 caracteres (*), sin incluir los símbolos <>.

(*) 31 caracteres si la opción de compatibilidad "Guardar métodos como Unicode" está deseleccionada. (ver **Página Compatibilidad**)

Ejemplos

```
<>vlProcesoID:=Current process
<>vsKey:=Char(KeyCode)
If(<>vtNombre#"")
```

Variables proceso

Designa una variable proceso utilizando su nombre (el cual no puede comenzar con los símbolos <> ni por el signo dólar *$*). El nombre de una variable proceso puede contener hasta 255 caracteres(*).

(*) 31 caracteres si la opción de compatibilidad "Guardar los métodos en Unicode" está deseleccionada. (ver **Página Compatibilidad**)

Ejemplos

```
<>vrGrandTotal:=Sum([Accounts]Amount)
If(bValidate=1)
   vsCurrentName:=""
```

Variables locales

Designa una variable local precediendo un signo dólar ($) a su nombre. Un nombre de variable local puede contener hasta 255 caracteres (*), sin incluir el signo dólar.

(*) 31 caracteres si la opción de compatibilidad "Guardar los métodos en Unicode" está deseleccionada. (ver **Página Compatibilidad**)

Ejemplos

```
For($vlRecord;1;100)
   If($vsTempVar="No")
      $vsMyString:="Hello there"
```

Arrays

Designa un array escribiendo su nombre, el cual es el nombre que pasó a un comando de declaración de array (tal como **ARRAY LONGINT**) cuando creó el Array. Los arrays son variables, y desde el punto de vista del alcance, como las variables, hay tres diferentes tipos de arrays:

- Arrays interproceso,
- Arrays proceso,
- Arrays locales.

Arrays interproceso
El nombre de un Array interproceso está precedido por los símbolos (<>) — un signo "menor que" seguido por un signo "mayor que".

Nota: esta sintaxis puede utilizarse en Windows y Macintosh. Además, únicamente en Macintosh, puede utilizar el carácter diamante (Opción-Mayús-V en teclado en castellano).

Un nombre de array interproceso puede contener hasta 255 caracteres (*), sin incluir los símbolos <>.

Ejemplos

```
ARRAY TEXT(<>attemas;Records in table([Temas]))
SORT ARRAY(<>asPalabrasClaves;>)
ARRAY INTEGER(<>aiGranArray;10000)
```

Arrays proceso
Designa un array proceso utilizando su nombre (el cual no puede comenzar con los símbolos <> ni con el signo dólar $). El nombre de un array proceso puede contener hasta 255 caracteres (*).

Ejemplos

```
ARRAY TEXT(atTemas;Records in table([Temas]))
SORT ARRAY(asPalabrasClave;>)
ARRAY INTEGER(aiGranArray;10000)
```

Arrays locales
El nombre de un array local está precedido por el signo dólar ($). El nombre de un Array local puede contener hasta 255 (*) caracteres, sin incluir el signo dólar.

Ejemplos

```
ARRAY TEXT($atSubjects;Records in table([Topics]))
SORT ARRAY($asKeywords;>)
ARRAY INTEGER($aiBigArray;10000)
```

(*) 31 caracteres si la opción de compatibilidad "Guardar los métodos en Unicode" está deseleccionada. (ver **Página Compatibilidad**)

Elementos de arrays
Se referencia un elemento de un array interproceso, proceso o local utilizando las llaves ({…}). El elemento referenciado se indica por una expresión numérica.

Ejemplos

```
   ` Direccionar un elemento de un array interproceso
If(<>asPalabrasClave{1}="Parar")
   <>atAsuntos{$vlElem}:=[Temas]Asunto
   $viPróximoValor:=<>aiGranArray{Size of array(<>aiGranArray)}

   ` Direccionar un elemento de un array proceso
   If(asPalabrasClave{1}="Parar")
      atAsuntos{$vlElem}:=[Temas]Asunto
      $viProximoValor:=aiGranArray{Size of array(aiGranArray)}

   ` Direccionar un elemento de un array local
      If($asPalabrasClave{1}="Parar")
```

```
$atAsuntos{$vlElem}:=[Temas]Asunto
$viProximotValor:=$aiGranArray{Size of array($aiGranArray)}
```

Elementos de arrays de dos dimensiones
Se referencia un elemento de un array de dos dimensiones utilizando un par de llaves ({…}). El elemento referenciado se indica por dos expresiones numéricas en dos pares de llaves.
Ejemplos

```
` Direccionar un elemento de un array interproceso de dos dimensiones
If(<>asPalabrasClave{$vlLineaSiguiente}{1}="Parar")
   <>atAsuntos{10}{$vlElem}:=[Temas]Asunto
   $viValorSiguiente:=<>aiGranArray{$vlSet}{Size of array(<>aiGranArray{$vlSet})}

` Direccionar un elemento de un array de proceso de dos dimensiones
   If(asPalabrasClave{$vlLineaSiguiente}{1}="Parar")
      atSubjects{10}{$vlElem}:=[Temas]Tema
      $viValorSiguiente:=aiGranArray{$vlSet}{Size of array(aiGranArray{$vlSet})}

` Direccionar un elemento de un array local de dos dimensiones
      If($asPalabrasClave{$vlNLineaSiguiente}{1}="Parar")
         $atAsuntos{10}{$vlElem}:=[Temas]Tema
         $viValorSiguiente:=$aiGranArray{$vlSet}{Size of array($aiGranArray{$vlSet})}
```

Atributos de objetos

Cuando se habilita la notación de objetos (ver **Página Compatibilidad**), se designa un atributo objeto (también llamado propiedad objeto) colocando un punto (".") entre el nombre del objeto (o atributo) y el nombre del atributo. Un nombre de atributo puede contener hasta 255 caracteres y distingue entre mayúsculas y minúsculas.
Ejemplos:

```
myObject.myAttribute:="10"
$value:=$clientObj.data.address.city
```

Nota: se aplican reglas adicionales a los nombres de atributos de objetos (deben ajustarse a la especificación ECMA Script). Para más información, consulte **Uso de la notación objeto**.

Formularios

Usted designa un formulario utilizando una expresión de tipo cadena alfanumérica que representa su nombre. El nombre de un formulario puede contener hasta 31 caracteres.
Ejemplos

```
FORM SET INPUT([Personas];"Entrada")
FORM SET OUTPUT([Personas];"Salida")
DIALOG([Deposito];"Caja de notas"+String($vlEtapa))
```

Objetos de formularios

Usted designa un objeto de formulario pasando su nombre como una cadena, precedido por el parámetro *. Un nombre de objeto puede contener hasta 255 bytes.

Ejemplo:

```
OBJECT SET FONT(*;"Binfo";"Times")
```

Ver también la sección **Propiedades de los objetos**.
Nota: no confunda objetos de formulario (botones, list boxes, variables que se pueden introducir, etc.) y los objetos del lenguaje 4D. Los objetos del lenguaje 4D se crean y manipulan a través de notación objeto o comandos dedicados. (ver **Objetos (Lenguaje)**)

Métodos

Designe un método (procedimiento o función usuario) utilizando su nombre. El nombre de un método puede contener hasta 31 caracteres.
Nota: un método que no devuelve un resultado también se llama un **procedimiento**. Un método que devuelve un resultado también se llama **función**.
Ejemplos

```
If(Nuevo cliente)
   BORRAR VALORES DUPLICADOS
```

```
APPLY TO SELECTION([Empleados];AUMENTAR SALARIOS)
```

Consejo: es una buena técnica de programación adoptar la misma convención de nombres que utiliza 4D para comandos integrados. Utilice caracteres en mayúsculas para los nombres de sus métodos; sin embargo si un método es una función, coloque en mayúsculas el primer carácter de su nombre. Al hacer esto, cuando reabra una base para mantenimiento después de unos meses, identificará si un método devuelve un resultado con sólo mirar su nombre en la ventana del Explorador.

Nota: cuando usted llama un método, simplemente digita su nombre. Sin embargo, algunos comandos integrados 4D, tales como **ON EVENT CALL**, así como también los comandos de plug-in, necesitan el nombre de un método como una cadena cuando se pasa un parámetro de tipo método:

Ejemplos

```
` Este comando espera un método (función) o fórmula
QUERY BY FORMULA([aTabla];Special query)
` Este comando espera un método (procedimiento) o fórmula
APPLY TO SELECTION([Empleados];AUMENTAR SALARIOS)
` Pero este comando espera un nombre de método
ON EVENT CALL("MANEJAR EVENTOS")
` Y este comando de plug-ins espera un nombre de método
WR ON ERROR("WR MANEJAR ERRORES")
```

Los métodos pueden captar parámetros (argumentos). Los parámetros se pasan al método entre paréntesis, siguiendo el nombre del método. Cada parámetro está separado del siguiente por un punto y coma (*;*). Los parámetros están disponibles dentro del método llamado como variables locales numeradas consecutivamente: *$1*, *$2*,..., *$n*. Además, varios parámetros consecutivos (y últimos) pueden ser direccionados con la sintaxis *${n}* donde n, expresión numérica, es el número del parámetro.

Dentro de una función, la variable local *$0* contiene el valor a devolver.

Ejemplos

```
` En ELIMINAR ESPACIOS $1 es un puntero al campo [Personas]Nombre
ELIMINAR ESPACIOS(->[Personas]Nombre)

` En Creador calc:
` - $1 es un numérico y es igual a 1
` - $2 es un numérico y es igual a 5
` - $3 es texto o cadena y es igual a "Súper"
` - El valor resultante se asigna a $0
$vsResult:=Creador calc(1;5;"Súper")

` En Botar:
` - Los tres parámetros son texto o cadena
` - Pueden ser direccionados como $1, $2 o $3
` - También pueden ser direccionados como, por ejemplo, ${$vlParam} donde $vlParam es 1, 2 o 3
` - El valor resultante se asigna a $0
vtClon:=Botar("es";"el";"él")
```

Comandos de plug-ins (procedimientos, funciones y áreas externas)

Usted designa un comando de plug-in utilizando su nombre como se definió en el plug-in. Un nombre de comando de plug-in puede contener hasta 31 caracteres.

Ejemplo

```
$error:=SMTP_From($smtp_id;"henry@gmail.com")
```

Conjuntos

Desde el punto de vista del alcance, hay dos tipos de conjuntos:

- Conjuntos interproceso
- Conjuntos proceso.

4D Server también incluye:

- Conjuntos clientes.

Conjuntos interproceso
Un conjunto es un conjunto interproceso si el nombre del conjunto está precedido por los símbolos (<>) — un signo "menor que" seguido por un signo "mayor que".

Nota: esta sintaxis puede utilizarse en Windows y Macintosh. Además únicamente en Macintosh, puede utilizar el carácter diamante (Opción-Mayús-V en teclado en castellano).

El nombre de un conjunto interproceso puede contener hasta 255 caracteres, sin incluir los símbolos <>.

Conjuntos proceso
Usted declara un conjunto proceso utilizando una expresión de tipo cadena que representa su nombre (el cual no puede comenzar con los símbolos <> o *$*). Un nombre de conjunto proceso puede contener hasta 255 caracteres.

Conjuntos cliente
El nombre de un conjunto cliente está precedido por el signo dólar (*$*). Un nombre de conjunto cliente puede contener hasta 255 caracteres, sin incluir el signo dólar.

Nota: Los conjuntos son administrados por el equipo servidor. En algunos casos, por razones especiales o de eficiencia, usted podría necesitar trabajar con conjuntos locales en el equipo cliente. Para hacerlo, utilice conjuntos cliente.

Ejemplos

```
  ` Conjuntos interproceso
USE SET("<>Registros borrados")
CREATE SET([Clientes];"<>Ordenes clientes")
If(Records in set("<>Seleccion"+String($i))>0)
  ` Conjuntos proceso
   USE SET("Registros borrados")
   CREATE SET([Clientes];"Ordenes clientes")
   If(Records in set("<>Seleccion"+String($i))>0)
  ` Conjuntos cliente
      USE SET("$Registros borrados")
      CREATE SET([Clientes];"$Ordenes clientes")
      If(Records in set("$Seleccion"+String($i))>0)
```

Selecciones temporales

Desde el punto de vista del alcance, hay dos tipos de selecciones temporales:

- Selecciones temporales interproceso
- Selecciones temporales proceso.

Selecciones temporales interproceso
Una selección temporal es una selección temporal interproceso si su nombre está precedido por los símbolos (<>) — un signo "menor que" seguido por un signo "mayor que".

Nota: esta sintaxis puede utilizarse en Windows y Macintosh. Además únicamente en Macintosh, puede utilizar el carácter diamante (Opción-Mayús-V en teclado en castellano).

El nombre de una selecciónt emporal interproceso puede contener hasta 255 caracteres, sin incluir los símbolos <>.

Selecciones temporales proceso
Usted declara una selección temporal proceso utilizando una expresión de tipo cadena que represente su nombre (la cual no puede comenzar con los símbolos <> o el signo dólar *$*). Un nombre de un selección temporal proceso puede contener hasta 255 caracteres.

Ejemplos

```
  ` Selección temporal interproceso
USE NAMED SELECTION([Clientes];"<>PorCodigopostal")
  ` Selección temporal proceso
USE NAMED SELECTION([Clientes];"PorCodigopostal")
```

Procesos

En versión monousuario, o Cliente/Servidor en el equipo cliente, hay dos tipos de procesos:

- Procesos globales
- Procesos locales.

Procesos globales
Usted declara un proceso global utilizando una expresión de tipo cadena que represente su nombre (la cual no puede comenzar con el signo dólar *$*). El nombre de un proceso puede contener hasta 255 caracteres.

Procesos locales
Usted declara un proceso local si el nombre del proceso está precedido por un signo dólar (*$*). El nombre de un proceso local puede contener hasta 255 caracteres, sin incluir el signo dólar.

Ejemplo

```
  ` Iniciar el proceso global "Añadir clientes"
$vlProcesoID:=New process("P_ADD_CUSTOMERS";48*1024;"Añadir clientes")
  ` Iniciar el proceso local "$Seguir Movimientos Ratón"
$vlProcesoID:=New process("P_MOUSE_SNIFFER";16*1024;"$Seguir movimientos del ratón")
```

Resumen de las convenciones de escritura

La siguiente tabla resume las principales convenciones escritura en los métodos.

Identificador	Long. max.	Ejemplo
Tabla	31	[Facturas]
Campo	31	[Empleados]Apellido
Variable/Array interproceso	<> + 255(*)	<>vlProcesoSiguienteID
Variable/Array proceso	255(*)	vsNombreActual
Variable/Array local	$ + 255(*)	$vlContadorLocal
Atributos de objetos	255	$o.*myAttribute*
Local Array	$ + 31	$atValues
Formulario	31	"Formulario Web personalizado"
Objeto de formulario	255	"MyButton"
Array interproceso	<> + 31	<>apTablas
Array proceso	31	asGenero
Array local	$ + 31	$atValores
Método	31	M_AÑADIR_CLIENTES
Comando de plug-in	31	WR INSERTAR TEXTO
Conjunto interproceso	<> + 255	"<>Registros a archivar"
Conjunto proceso	255	"Registros actuales seleccionados"
Conjunto cliente	$ + 255	"$Temas anteriores"
Selección temporal	255	"Empleados de A a Z"
Selección temporal interproceso	<> + 225	"<>Empleados de Z a A"
Proceso local	$ + 255	"$Seguir Eventos"
Proceso global	255	"P_MODULO_FACTURAS"
Semáforo	255	"mysemaphore"

(*) 31 caracteres si la opción de compatibilidad "Guardar métodos como Unicode" no está seleccionada. (ver **Página Compatibilidad**)

Nota: si se utilizan caracteres no romanos en los nombres de los identificadores, su tamaño máximo puede ser menor.)

Resolver conflictos de nombres

Asegúrese de utilizar nombres únicos para los diferentes elementos de su base de datos. Si un objeto determinado tiene el mismo nombre que otro objeto de un tipo diferente (por ejemplo, si un campo se denomina Person y una variable también se denomina Person), 4D utiliza un sistema de prioridad.

4D identifica los nombres utilizados en lo procedimientos en el siguiente orden:

1. Campos
2. Comandos
3. Métodos
4. Rutinas de plug-ins
5. Constantes predefinidas
6. Variables.

Por ejemplo, 4D tiene un comando integrado llamado **Date**. Si llama a un método Date, 4D lo reconocerá como el comando integrado **Date**, y no como su método. Esto puede evitar que llame a su método. Si, a pesar de esto, usted llama a un campo *"Date"*, 4D tratará de utilizar su campo en lugar del comando **Date**.

Condiciones y bucles

Sin importar la simplicidad o complejidad de un método, usted utilizará siempre uno o más de tres tipos de estructuras de programación. Las estructuras de programación controlan el flujo de ejecución, si las instrucciones en un método son ejecutadas y en qué orden. Hay tres tipos de estructuras:

- Secuenciales
- Condicionales
- Bucles

El lenguaje de 4D contiene instrucciones que permiten controlar cada una de estas estructuras.

Estructuras secuenciales

La estructura secuencial es una estructura simple, lineal. Una secuencia es una serie de instrucciones que 4D ejecuta una tras otra, de la primera a la última. Por ejemplo:

```
OUTPUT FORM([Personas];"Listar")
ALL RECORDS([Personas])
DISPLAY SELECTION([Personas])
```

Una instrucción de una línea, frecuentemente utilizada por los métodos de objeto, es el caso más simple de estructura secuencial. Por ejemplo:

```
[Personas]Apellido:=Uppercase([Personas]Apellido)
```

Nota: las palabras clave Begin SQL / End SQL son usadas para delimitar estrcuturas secuenciales a ejecutar por el motor SQL de 4D. Para mayor información, consulta la descripción de estas palabras claves.

Estructuras condicionales

Una estructura condicional permite a los métodos probar una condición y tomar rutas alternativas, dependiendo del resultado. La condición es una expresión booleana, una expresión que evalúa TRUE o FALSE. Una de las estructuras condicionales es la estructura **If...Else...End if**, la cual direcciona el flujo del programa a través de una o dos rutas. La otra estructura condicional es la estructura **Case of...Else...End case**, la cual direcciona el flujo del programa a una de muchas rutas.

Estructuras bucle

Cuando escribe métodos, es muy común encontrar que debe repetir una secuencia de instrucciones un cierto número de veces. Para manejar esta necesidad, el lenguaje ofrece tres estructuras bucle:

- **While...End while**
- **Repeat...Until**
- **For...End for**
- **For each...End for each**

Los bucles se controlan de dos maneras: o se repiten hasta que se cumpla una condición, o se repiten un número específico de veces. Cada estructura bucle puede utilizarse de cualquiera de las dos formas, pero los bucles While y Repeat son más apropiados para ser repetidos hasta que se cumpla una condición, y los bucles For son más apropiados para ser repetidos un número específico de veces. **For each...End for each** permite mezclar en ambos sentidos y está diseñado para recorrer dentro de objetos y colecciones.

Nota: 4D le permite anidar estructuras de programación (If/While/For/Case of/Repeat) hasta una "profundidad" de 512 niveles.

If...Else...End if

La sintaxis de la estructura condicional **If...Else...End if** es la siguiente:

```
If(Expresion_Booleana)
    instrucción(es)
Else
    instrucción(es)
End if
```

Observe que la parte **Else** es opcional; puede escribir:

```
If(Expresion_Booleana)
    instrucción(es)
End if
```

La estructura **If...Else...End if** permite a su método elegir entre dos acciones, dependiendo de si una prueba (una expresión Booleana) es VERDADERA (TRUE) o FALSA (FALSE).

Cuando la expresión booleana es TRUE, se ejecutan las instrucciones que siguen inmediatamente después de la prueba. Si la expresión booleana es FALSE, las instrucciones siguientes a la instrucción **Else** son ejecutadas. La instrucción **Else** es opcional; si omite **Else**, continua la ejecución con la primera instrucción (si la hay) después de **End if**.

Note que la expresión Booleana siempre se evalúa completamente. Considere en particular la siguiente prueba:

```
If(MethodA & MethodB)
    ...
End if
```

La expresión es TRUE sólo si ambos métodos son TRUE. Sin embargo, incluso si ***MethodA*** devuelve FALSE, 4D 4D todavía evaluará ***MethodB***, lo que es una pérdida inútil de tiempo. En este caso, es más interesante utilizar una estructura como:

```
If(MethodA)
    If(MethodB)
        ...
    End if
End if
```

El resultado es similar y ***MethodB*** se evalúa sólo si es necesario.

Ejemplo

```
` Pedir al usuario introducir un nombre
$Encontrar:=Request(Digite un nombre)
If(OK=1)
    QUERY([Personas];[Personas]Apellido=$Encontrar)
Else
    ALERT("Usted no introdujo ningún nombre.")
End if
```

Consejo: la bifurcación se puede realizar sin ejecutar instrucciones en un caso o en el otro. Cuando desarrolla un algoritmo o una aplicación especializada, nada le impide escribir:

```
If(Expresion_Booleana)
Else
    instrucción(es)
End if
```

o:

```
If(Expresion_Booleana)
    instrucción(es)
Else
End if
```

Case of...Else...End case

La sintaxis de la estructura condicional **Case of...Else...End case** es la siguiente:

```
Case of
   :(Expresion_Booleana)
      instrucción(es)
   :(Expresion_Booleana)
      instrucción(es)
      .
      .
      .

   :(Expresion_Booleana)
      instrucción(es)
   Else
      instrucción(es)
End case
```

Observe que la parte **Else** es opcional; puede escribir:

```
Case of
   :(Expresion_Booleana)
      instrucción(es)
   :(Expresion_Booleana)
      instrucción(es)
      .
      .
      .

   :(Expresion_Booleana)
      instrucción(es)
End case
```

Como en la estructura **If...Else...End if**, la estructura **Case of...Else...End case** también le permite a su método elegir entre acciones alternativas. A diferencia de la estructura **If...Else...End if**, la estructura **Case of...Else...End case** puede probar un número ilimitado de expresiones booleanas y ejecutar la secuencia de instrucción correspondiente al valor TRUE.

Cada expresión booleana comienza con dos puntos (*:*). Esta combinación de dos puntos y una expresión booleana se llama caso. Por ejemplo, la siguiente línea es un caso:

```
:(bValidar=1)
```

Sólo las instrucciones después del primer caso TRUE (y hasta el siguiente caso) serán ejecutadas. Si ninguno de los casos es TRUE, ninguna de las instrucciones se ejecutará (si no está incluida la parte **Else**).

Puede incluir una instrucción **Else** después del último caso. Si todos los casos son FALSE, las instrucciones después de **Else** serán ejecutadas.

Ejemplo

Este ejemplo prueba una variable numérica y muestra una caja de diálogo de alerta con una palabra en ella:

```
Case of
   :(vResult=1)  ` Prueba si el número es 1
      ALERT("Uno.")  ` Si es 1, muestra una alerta
   :(vResult=2)  ` Prueba si el número es 2
      ALERT("Dos.")  ` Si es 2, muestra una alerta
   :(vResult=3)  ` Prueba si el número es 3
      ALERT("Tres.")  ` Si es 3, muestra una alerta
   Else  ` Si no es 1, 2, o 3, muestra una alerta
      ALERT("No es ni uno, ni dos, ni tres.")
End case
```

Para comparar, esta es la versión con **If...Else...End if** del mismo método:

```
If(vResult=1)  ` Prueba si el número es 1
   ALERT("Uno.")  ` Si es 1, muestra una alerta
```

```
Else
   If(vResult=2)  ` Prueba si el número es 2
      ALERT("Dos.")  ` Si es 2, muestra una alerta
   Else
      If(vResult=3)  ` Prueba si el número es 3
         ALERT("Tres.")  ` Si es 3, muestra una alerta
      Else  ` Si no es 1, 2, o 3, muestra una alerta
         ALERT("No es ni uno, ni dos, ni tres.")
      End if
   End if
End if
```

Recuerde que con una estructura **Case of...Else...End case**, sólo el primer caso TRUE se ejecuta. Incluso si dos o mas casos son TRUE, sólo las instrucciones después del primer caso TRUE se ejecutarán.

Por lo tanto, cuando quiera implementar pruebas jerárquicas, debe asegurarse de que las instrucciones de condición que están más abajo en el esquema jerárquico aparezcan primero en la secuencia de prueba. Por ejemplo, la prueba de la presencia de la condicion1 cubre la prueba de la presencia de la condicion1&condicion2 y por lo tanto debe estar ubicada al final en la secuencia de prueba. Por ejemplo, el siguiente código nunca verá su última condición detectada:

```
Case of
   :(vResult=1)
      ...  `instrucción(es)
   :((vResult=1) & (vCondicion#2))  `este caso nunca será detectado
      ...  `instrucción(es)
End case
.
```

En el código anterior, la presencia de la segunda condición no se detecta ya que la prueba "vResult=1" desvía el código anter de realizar más pruebas. Para que el código funcione correctamente, puede escribirlo de la siguiente manera:

```
Case of
   :((vResult=1) & (vCondicion#2))  `este caso será detectado primero
      ...  `instrucción(es)
   :(vResult=1)
      ...  `instrucción(es)
End case
.
```

Igualmente, si quiere implementar pruebas jerárquicas, podría considerar utilizar código jerárquico.

Consejo: la bifurcación puede realizarse sin instrucciones a ejecutar en un caso u otro. Cuando desarrolla un algoritmo o una aplicación especializada, nada evita que escriba:

```
Case of
   :(Expresion_Booleana)
   :(Expresion_Booleana)

      .
      .
      .

   :(Expresion_Booleana)
      instrucción(es)
   Else
      instrucción(es)
End case
```

o:

```
Case of
   :(Expresion_Booleana)
   :(Expresion_Booleana)
      instrucción(es)
      .
      .
      .

   :(Expresion_Booleana)
      instrucción(es)
   Else
End case
```

o:

```
Case of
    Else
        instrucción(es)
End case
```

While...End while

La sintaxis formal de la estructura de control de flujo **While...End while** es:

```
While(Expresion_Booleana)
    instrucción(es)
End while
```

Un bucle **While...End while** ejecuta las instrucciones en el bucle mientras que la expresión booleana sea TRUE. Prueba la expresión booleana al comienzo del bucle y no entra al bucle si la expresión es FALSE.

Es común inicializar el valor probado en la expresión booleana antes de introducir el bucle **While...End while**. Inicializar el valor significa definirlo para algo apropiado, generalmente para que la expresión booleana sea verdadera y que **While...End while** ejecute el bucle.

La expresión booleana debe ser definida por algo dentro del bucle o si no el bucle continuará indefinidamente. El siguiente bucle continúa indefinidamente porque **Infinito** siempre es TRUE:

```
Infinito:=True
While(Infinito)
End while
```

Si se encuentra en una situación así, donde un método se ejecuta de manera descontrolada, puede utilizar la funciones de ejecución paso a paso para detener el bucle y encontrar el problema. Para mayor información sobre ejecución paso a paso de un método, consulte la sección **Depurador**.

Ejemplo

```
CONFIRM("¿Añadir un nuevo registro?") ` ¿Quiere el usuario añadir un registro?
While(OK=1) ` Bucle hasta que el usuario quiera
    ADD RECORD([aTabla]) ` Añadir un nuevo registro
End while ` El bucle siempre termina con End while
```

En este ejemplo, el valor de la variable del sistema OK está definido por el comando **CONFIRM** antes de que comience el bucle. Si el usuario hace clic en el botón OK en la caja de diálogo de confirmación, la variable OK toma el valor 1 y el bucle comienza. De lo contrario, la variable OK toma el valor 0 y se ignora el bucle. Una vez comienza el bucle, el comando **ADD RECORD** permite continuar ejecutando el bucle porque hace que la variable OK tome el valor 1 cuando el usuario guarda el registro. Cuando el usuario cancela (no guarda) el último registro, la variable OK toma el valor 0 y el bucle se detiene.

Repeat...Until

La sintaxis formal de la estructura de control de flujo **Repeat...Until** es:

```
Repeat
    instrucción(es)
Until(Expresion_Booleana)
```

Un bucle **Repeat...Until** loop es similar al bucle **While...End while**, excepto que prueba la expresión booleana después de la ejecución del bucle en lugar de antes. De esta forma, el bucle **Repeat...Until** siempre se ejecuta por lo menos una vez, mientras que si la expresión boolena es inicialmente falsa, el bucle **While...End while** no se ejecuta.

La otra diferencia con el bucle **Repeat...Until** es que el bucle continua hasta que la expresión booleana sea TRUE.

Ejemplo

Compare el siguiente ejemplo con el ejemplo para el bucle **While...End while**. Observe que no es necesario inicializar la expresión booleana, no hay comando **CONFIRM** para inicializar la variable OK.

```
Repeat
    ADD RECORD([aTabla])
Until(OK=0)
```

For...End for

La sintaxis formal de la estructura de flujo de control **For...End for** es:

```
For(Variable_Contador;Expresion_Inicio;Expresion_Fin{;Expresion_Incremento})
   instrucción(es)
End for
```

El bucle **For...End for** es un bucle controlado por una variable contador:

- La variable contador **Variable_Contador** es una variable numérica (Real, Entero, o Entero largo) que el bucle **For...End for** inicializa en el valor especificado por **Expresion_Inicio**.
- Cada vez que el bucle se ejecuta, el valor del contador se incrementa en el valor especificado por el parámetro opcional **Expresion_Incremento**. Si no especifica **Expresion_Incremento**, la variable contador se incrementará por defecto en uno (*1*).
- Cuando el contador alcanza el valor de **Expresion_Fin** , el bucle se detiene.

Importante: las expresiones numéricasExpresion_Inicio,Expresion_Fin yExpresion_Incremento son evaluadas una sola vez al comienzo del bucle. Si estas expresiones son variables, su modificación al **interior** del bucle **no** afectará el bucle.
Tip: sin embargo, con fines especiales, puede modificar el valor de la variableVariable_Contador
dentro del bucle; esto afectará el bucle.

- Generalmente **Expresion_Inicio** es menor que **Expresion_Fin** .
- Si **Expresion_Inicio** y **Expresion_Fin** son iguales, el bucle será ejecutado sólo una vez.
- Si **Expresion_Inicio** es mayor que **Expresion_Fin,** el bucle no se ejecutará a menos de que **Expresion_Incremento** sea negativo. Ver ejemplos.

Ejemplos básicos

1. El siguiente ejemplo ejecuta 100 iteraciones:

```
For(vContador;1;100)
  ` Hacer algo
End for
```

2. El siguiente ejemplo va a través de todos los elementos del Array *unArray*:

```
For($vlElem;1;Size of array(unArray))
  ` Hacer algo con el elemento
   unArray{$vlElem}:=...
End for
```

3. El siguiente ejemplo va a través de todos los caracteres del texto *vtTexto*:

```
For($vlCar;1;Length(vtTexto))
  ` Hacer algo con el carácter si es un TAB
   If(Character code(vtTexto[[$vlCar]])=Tab)
  ` ...
   End if
End for
```

4. El siguiente ejemplo va a través de los registros seleccionados para la tabla *[unaTabla]*:

```
FIRST RECORD([unaTabla])
For($vlRegistro;1;Records in selection([unaTabla]))
  ` Hacer algo con el registro
   SEND RECORD([unaTabla])
  ` ...
  ` Ir al siguiente registro
   NEXT RECORD([unaTabla])
End for
```

La mayoría de los bucles **For...End for** que usted escriba en sus bases se verán como los que presentamos en estos ejemplos.

Disminuir la variable contador

En algunos casos, podría querer tener un bucle cuyo contador disminuya en lugar de aumentar. Para hacer esto,Expresion_Inicio debe ser mayor que **Expresion_Fin** y **Expresion_Incremento** debe ser negativo. Los siguientes ejemplos efectúan las mismas tareas que los ejemplos anteriores, pero en orden inverso:

5. El siguiente ejemplo ejecuta 100 iteraciones:

```
For(vCounter;100;1;-1)
  ` Hacer algo
End for
```

6. El siguiente ejemplo va a través de todos los elementos del Array *unArray*:

```
For($vlElem;Size of array(anArray);1;-1)
  `Hacer algo con el elemento
    unArray{$vlElem}:=...
End for
```

7. El siguiente ejemplo va a través de todos los caracteres de texto *vtTexto*:

```
For($vlCar;Length(vtTexto);1;-1)
  ` Hacer algo con el carácter si es un TAB
    If(ASCII(vtTexto[[$vlCar]])=Tab)
  ` ...
    End if
End for
```

8. El siguiente ejemplo va a través de los registros seleccionados de la tabla *[unaTabla]*:

```
LAST RECORD([unaTabla])
For($vlRecord;Records in selection([unaTabla]);1;-1)
  ` Hacer algo con el registro
    SEND RECORD([unaTabla])
  ` ...
  ` Ir al registro anterior
    PREVIOUS RECORD([unaTabla])
End for
```

Incrementar la variable contador en más de uno

Si necesita esto, puede utilizar en **Expresion_Incremento** un valor (positivo o negativo) cuyo valor absoluto sera mayor que uno.

9. El siguiente bucle se dirigue sólo a los elementos pares del Array unArray:

```
For($vlElem;2;((Size of array(unArray)+1)\2)*2;2)
  ` Hacer algo con los elementos #2,#4...#2n
    unArray{$vlElem}:=...
End for
```

Observe que la expresión *((Size of Array(unArray)+1)\2)*2* trata los arrays pares e impares.

Salir de un bucle cambiando la variable contador

En algunos casos, usted podría querer ejecutar un bucle un cierto número de veces, pero luego salir del bucle cuando otra condición sea TRUE. Para hacer esto, puede probar esta condición dentro del bucle y si es TRUE, explicitamente definir la variable contador en un valor superior al valor de la expresión final.

10. En el siguiente ejemplo, se efectúa un bucle entre los registros de una selección hasta llegar al final de la selección o hasta que la variable interproceso *<>vbWeStop*, inicialmente definida como FALSE, se vuelva TRUE. Esta variable se maneja por un método de proyecto **ON EVENT CALL** que le permite interrumpir la operación:

```
<>vbWeStop:=False
ON EVENT CALL("GESTION STOP")
  ` GESTION STOP define <>vbStop como True si las teclas Ctrl-punto (Windows) o Cmd-Punto (Macintosh) son presionadas
$vlNumRegistros:=Records in selection([unaTabla])
FIRST RECORD([unaTabla])
For($vlRegistro;1;$vlNumRegistros)
  ` Hacer algo con el registro
    SEND RECORD([unaTabla])
  ` ...
  ` Ir al siguiente registro
    If(<>vbWeStop)
        $vlRecord:=$vlNbRecords+1  ` Forzar a la variable contador a salir del bucle
    Else
```

```
        NEXT RECORD([unaTabla])
    End if
End for
ON EVENT CALL("")
If(<>vbStop)
    ALERT("La operación ha sido interrumpida.")
Else
    ALERT("La operación ha terminado con éxito.")
End if
```

Comparación de estructuras de bucle

Regresemos al primer ejemplo **For...End for**:
El siguiente ejemplo ejecuta 100 iteraciones:

```
For(vCounter;1;100)
  ` Hacer algo
End for
```

Es interesante ver cómo los bucles **While...End while** y **Repeat...Until** realizan la misma acción.
Este es el bucle **While...End while** equivalente :

```
$i :=1  ` Inicialización del contador
While($i<=100)  ` Bucle 100 veces
  ` Hacer algo
    $i :=$i +1  ` Hay que incrementar el contador
End while
```

Este es el bucle **Repeat...Until** equivalente:

```
$i :=1  ` Inicialización del contador
Repeat
  ` Hacer algo
    $i :=$i +1  ` Hay que incrementar el contador
Until($i=100)  ` Bucle 100 veces
```

Consejo: el bucle **For...End for** es generalmente más rápido que los bucles **While...End while** y **Repeat...Until** , porque 4D prueba la condición internamente para cada ciclo del bucle e incrementa el contador. Por lo tanto, utilice la estructura **For...End for** cada vez que sea posible.

Optimizar la ejecución de For...End for

Puede utilizar como contador una variable interproceso, proceso o local de tipo Real, Entero, y Entero largo. Para bucles largos, especialmente en modo compilado, utilice variables locales de tipo entero largo.
11. Este es un ejemplo:

```
C_LONGINT($vlContador)  ` uso de una variable local de tipo entero largo
For($vlContador;1;10000)
  ` Hacer algo
End for
```

Estructuras For...End for de bucles anidados

Usted puede anidar tantas estructuras de control como necesite. Esto incluye anidamiento de bucles **For...End for**. Para evitar errores, asegúrese de utilizar una variable contador diferente para cada estructura de bucle
Estos son dos ejemplos:
12. El siguiente ejemplo va a través de todos los elementos de un Array de dos dimensiones:

```
For($vlElem;1;Size of array(unArray))
  ` ...
  ` Hacer algo con la fila
  ` ...
    For($vlSubElem;1;Size of array(unArray{$vlElem}))
  ` Hacer algo con el elemento
        unArray{$vlElem}{$vlSubElem}:=...
    End for
End for
```

13. El siguiente ejemplo construye un Array de punteros para todos los campos de tipo Fecha presentes en la base:

```
ARRAY POINTER($apCamposFecha;0)
$vlElem:=0
For($vlTabla;1;Count table)
    For($vlCampo;1;Count fields($vlTabla))
        $vpCampo:=Field($vlTabla;$vlCampo)
        If(Type($vpCampo->)=Is date)
            $vlElem:=$vlElem+1
            INSERT IN ARRAY($apCamposFecha;$vlElem)
            $apCamposFecha{$vlElem}:=$vpCampo
        End if
    End if
End for
End if
End for
```

For each...End for each

La sintaxis formal de la estructura repetitiva o bucle **For each...End for each** es la siguiente:

```
For each(Current_Item;Expression{;begin{;end}}){Until|While}(Boolean_Expression)}
   statement(s)
End for each
```

La estructura **For each...End for each** ejecuta el ciclo de instrucciones definidas para cada *Elemento_actual* de *Expresion*. El tipo *Elemento_actual* depende del tipo de *Expresion*. El bucle **For each...End for each** puede iterar a través de tres tipos de *Expresion*:

- **colecciones**: bucle en cada elemento de la colección,
- **entity selections**: bucle a través de cada entidad,
- **objetos**: bucle en cada propiedad del objeto.

La siguiente tabla compara los tres tipos de For each...End for each:

	Bucle en colecciones	**Bucle en entity selections**	**Bucle en objetos**
Tipo *Current_Item*	Variable del mismo tipo que los elementos de la colección	Entity	Variable texto
Tipo *Expression*	Colección (con elementos del mismo tipo)	Selección de entidad	Objeto
Número de bucles (por defecto)	Número de elementos de la colección	Número de entidades en la selección	Número de propiedades de objetos
Soporte de Parámetros inicio/fin	Sí	Sí	No

- El número de bucles se evalúa al inicio y no cambiará durante el procesamiento. Por lo general, no se recomienda agregar o eliminar elementos durante el ciclo, ya que puede dar como resultado iteraciones faltantes o redundantes.
- Por defecto, las *instrucciones* adjuntas se ejecutan para cada valor en *Expresion*. Sin embargo, es posible salir del bucle probando una condición ya sea al comienzo del bucle (**While**) o al final del bucle (**Until**).
- Los parámetros opcionales *inicio* y *fin* se pueden usar con colecciones y selecciones de entidades para definir los límites del bucle.

Bucles en las colecciones

Cuando **For each...End for each** se usa con una *Expresion* de tipo *Collection*, el parámetro *Elemento_actual* es una variable del mismo tipo que los elementos de la colección. Por defecto, el número de bucles se basa en la cantidad de elementos de la colección.

La colección debe contener solo elementos del mismo tipo; de lo contrario, se devolverá un error tan pronto como a la variable *Elemento_actual* se le asigne el primer valor de tipo diferente.

En cada iteración de bucle, la variable *Elemento_actual* se llena automáticamente con el elemento coincidente de la colección. Los siguientes puntos deben tenerse en cuenta:

- Si la variable *Elemento_actual* es de tipo objeto o colección (es decir, si *Expresion* es una colección de objetos o de colecciones), modificar esta variable modificará automáticamente el elemento correspondiente de la colección (porque los objetos y las colecciones comparten las mismas referencias). Si la variable es de tipo escalar, solo se modificará la variable.
- La variable *Elemento_actual* debe ser del mismo tipo que los elementos de la colección. Si algún elemento de la colección no es del mismo tipo que la variable, se genera un error y el ciclo se detiene.
- Si la colección contiene elementos con un valor **Null**, se generará un error si el tipo de variable *Elemento_actual* no admite valores **Null** (como las variables entero largo).

Ejemplo

Usted desea calcular algunas estadísticas para una colección de números:

```
C_COLLECTION($nums)
$nums:=New collection(10;5001;6665;33;1;42;7850)
C_LONGINT($item;$vEven;$vOdd;$vUnder;$vOver)
For each($item;$nums)
   If($item%2=0)
      $vEven:=$vEven+1
   Else
      $vOdd:=$vOdd+1
   End if
   Case of
      :($item<5000)
         $vUnder:=$vUnder+1
      :($item>6000)
         $vOver:=$vOver+1
```

```
    End case
End for each
 //$vEven=3, $vOdd=4
 //$vUnder=4,$vOver=2
```

Bucles en las entity selections

Cuando **For each...End for each** se utiliza con una *Expresion* de tipo *Entity selection*, el parámetro *Elemento_actual* contiene la entidad que se procesa actualmente.

El número de bucles se basa en el número de entidades en la entity selection. En cada iteración del bucle, el parámetro *Elemento_actual* recibe automáticamente la entidad que está siendo procesada.

Nota: si la entity selection contiene una entidad que fue eliminada mientras tanto por otro proceso, se omite automáticamente durante el ciclo.

Tenga en cuenta que cualquier modificación que se aplique a la entidad actual se debe guardar explícitamente utilizando **entity.save( )**.

Ejemplo

Desea elevar el salario de todos los empleados británicos en una entity selection:

```
C_OBJECT(emp)
For each(emp;ds.Employees.query("country='UK'"))
    emp.salary:=emp.salary*1,03
    emp.save()
End for each
```

Bucles en las propiedades de objetos

Cuando **For each...End for each** se usa con una *Expresion* del tipo *Objeto*, el parámetro *Elemento_actual* es una variable texto que se completa automáticamente con el nombre de la propiedad actualmente procesada.
Las propiedades del objeto se procesan según su orden de creación. Durante el ciclo, las propiedades pueden agregarse o eliminarse del objeto, sin modificar el número de bucles que permanecerán basados en el número original de propiedades del objeto.

Ejemplo

Desea cambiar los nombres a mayúsculas en el siguiente objeto:

```
{    "firstname": "gregory",    "lastname": "badikora",    "age": 20 }
```

Puedo escribir:

```
For each(property;vObject)
    If(Value type(vObject[property])=Is text)
        vObject[property]:=Uppercase(vObject[property])
    End if
End for each
```

```
{    "firstname": "GREGORY",    "lastname": "BADIKORA",    "age": 20 }
```

Parámetros inicio/fin

Puede definir límites para la iteración utilizando los parámetros opcionales *inicio* y *fin*.

Nota: los parámetros *inicio* y *fin* solo se pueden usar en iteraciones a través de colecciones y de las entity selections (se ignoran en las propiedades de objetos).

- En el parámetro *inicio*, pase la posición del elemento en *Expresion* en donde comenzar la iteración (se incluye *inicio*).
- En el parámetro *fin*, también puede pasar la posición del elemento en *Expresion* en donde detener la iteración (se incluye *fin*).

Si *fin* se omite o si *fin* es mayor que el número de elementos en *Expresion*, los elementos se iteran desde *inicio* hasta el último (incluido).

Si los parámetros *inicio* y *fin* son valores positivos, representan las posiciones reales de los elementos en *Expresion*.

Si *inicio* es un valor negativo, se vuelve a calcular como *inicio:=inicio+Tamaño expresion* (se considera como el desplazamiento desde el final de *Expresion*). Si el valor calculado es negativo, *inicio* toma el valor 0.
Nota: incluso si *inicio* es negativo, la iteración se sigue realizando en el orden estándar.
Si *fin* es un valor negativo, se vuelve a calcular como *fin:=fin +Tamaño expresion*

Por ejemplo:

- una colección contiene 10 elementos (numerados de 0 a 9)
- inicio=-4 -> inicio=-4+10=6 -> la iteración comienza en el 6to elemento (#5)
- fin=-2 -> fin=-2+10=8 -> la iteración se detiene antes del 8° elemento (#7), es decir, en el 7mo elemento.

Ejemplo

```
C_COLLECTION($col;$col2)
$col:=New collection("a";"b";"c";"d";"e")
$col2:=New collection(1;2;3)
C_TEXT($item)
For each($item;$col;0;3)
   $col2.push($item)
End for each
 //$col2=[1,2,3,"a","b","c"]
For each($item;$col;-2;-1)
   $col2.push($item)
End for each
 //$col2=[1,2,3,"a","b","c","d"]
```

Condiciones Until y While

Puede controlar la ejecución de **For each...End for each** agregando una condición **Until** o **While** al bucle. Cuando una instrucción **Until** (condición) o **While** (condicion) está asociada al bucle, la iteración se detendrá tan pronto como la *condicion* se evalúe como **true**.

Puede pasar cualquiera de las palabras claves según sus necesidades:

- La condición **Until** se prueba al final de cada iteración, por lo que si *Expresion* no está vacía o null, el bucle se ejecutará al menos una vez.
- La condición **While** se prueba al comienzo de cada iteración, por lo que de acuerdo con el resultado de la condición, es posible que el bucle no se ejecute en absoluto.

Ejemplo

```
$colNum:=New collection(1;2;3;4;5;6;7;8;9;10)

$total:=0
For each($num;$colNum)While($total<30)  //probado al inicio
   $total:=$total+$num
End for each
ALERT(String($total))  //$total = 36 (1+2+3+4+5+6+7+8)

$total:=1000
For each($num;$colNum)Until($total>30)  //probado al final
   $total:=$total+$num
End for each
ALERT(String($total))  //$total = 1001 (1000+1)
```

Colecciones compartidas y objetos compartidos

El bucle **For each...End for each** se puede usar en una colección compartida o en un objeto compartido.

Si su código necesita modificar uno o más elementos de la colección o de las propiedades del objeto, debe usar las palabras claves **Use...End use**. Dependiendo de sus necesidades, puede llamar a las palabras claves **Use...End use**:

- antes de ingresar al bucle, si los elementos se deben modificar juntos por razones de integridad, o
- dentro del ciclo cuando solo se necesitan modificar algunos elementos/propiedades y no se requiere gestión de integridad.

Use...End use

La sintaxis formal de la estructura **Use...End use** es:

```
Use(Shared_object_or_Shared_collection)
   statement(s)
End use
```

La estructura **Use...End use** define una secuencia de instrucciones que ejecutará tareas en el parámetro *Shared_object_or_Shared_collection* bajo la protección de un semáforo interno. *Shared_object_or_Shared_collection* puede ser cualquier objeto compartido o colección compartida válidos.

Los objetos compartidos y las colecciones compartidas están diseñados para permitir la comunicación entre procesos, en particular, **Procesos 4D apropiativos**. Se pueden pasar por referencia como parámetros de un proceso a otro. Para información detallada sobre objetos compartidos o colecciones compartidas, consulte la página **Objetos y colecciones compartidos**. Las modificaciones circundantes en objetos compartidos o colecciones compartidas por las palabras claves **Use...End use** son obligatorias para evitar el acceso concurrente entre procesos.

- Una vez que la línea **Use** se ejecuta con éxito, todas las propiedades/elementos *Shared_object_or_Shared_collection* bloquean para el resto del proceso en el acceso de escritura hasta que se ejecuta la línea **End use** correspondiente.
- La secuencia *instruccion(es)* puede ejecutar cualquier modificación en las propiedades/elementos *Shared_object_or_Shared_collection* sin riesgo de acceso concurrente.
- Si se agrega otro objeto o colección como propiedad del parámetro *Shared_object_or_Shared_collection*, se conectan dentro del mismo **grupo compartido** ver **Utilizar objetos o colecciones compartidos**).
- Si otro proceso intenta acceder a una de las propiedades *Shared_object_or_Shared_collection* o a las propiedades conectadas mientras se está ejecutando una secuencia **Use...End use**, se pone automáticamente en espera y espera hasta que finalice la secuencia actual.
- La línea **End use** desbloquea las propiedades *Shared_object_or_Shared_collection* y todos los objetos que comparten el mismo identificador de *bloqueo*.
- Varias estructuras **Use...End use** se pueden anidar en el código 4D. En ese caso, todos los bloqueos se apilan y las propiedades/elementos se liberarán solo cuando se ejecute la última llamada **End use**.

Nota: si una función miembro de una colección (ver **Member functions**) modifica una colección compartida, se llama automáticamente a un **Use** interno para esta colección compartida mientras se ejecuta la función.

Métodos

Para que los comandos, operadores y otras partes del lenguaje funcionen, usted los pone en métodos. Hay varias clases de métodos: métodos de objeto, métodos de formulario, métodos de tabla (Triggers), Métodos de proyecto, y métodos de base. Esta sección describe las características comunes de todos los tipos de métodos.

Un método está compuesto de **instrucciones**; cada instrucción consiste de una línea en el método. Una instrucción realiza una acción, que puede ser simple o compleja. Aunque una instrucción siempre es una línea, la línea puede ser tan larga como sea necesaria (hasta 32 000 caracteres, lo cual es suficiente para la mayoría de las tareas).

Por ejemplo, la siguiente línea es una instrucción que añadirá un nuevo registro a la tabla [Personas]:

```
ADD RECORD([Personas])
```

Un método también contiene condiciones y **bucles** que controlan el flujo de ejecución. Para mayor información sobre estructuras de programación, consulte la sección **Condiciones y bucles**.

Nota: el tamaño máximo de un método está limitado a 2 GB de texto o 32 000 líneas de instrucciones. Más allá de estos límites, aparece un mensaje de advertencia, indicando que las líneas extras no aparecerán.

Tipos de métodos

Hay cinco tipos de métodos en 4D:

- **Métodos de objeto**: un método de objeto es una propiedad de un objeto. Generalmente es un método corto asociado con un objeto activo de formulario. Usualmente los métodos de objeto "administran" el objeto mientras el formulario se visualiza o imprime. Usted no puede llamar un método de objeto, 4D lo llama automáticamente cuando un evento atañe al objeto al cual el método está adjunto.
- **Métodos de formulario**: un método de formulario es una propiedad de un formulario. Puede utilizar un método de formulario para administrar datos y objetos, pero generalmente es más sencillo y más eficiente utilizar un método de objeto para estos propósitos. Usted no puede llamar un método de formulario, 4D lo llama automáticamente cuando un evento atañe al formulario al cual el método está adjunto.

Para mayor información sobre métodos de objeto y métodos de formulario, consulte el Manual de Diseño como también la sección **Eventos de formulario**.

- **Métodos tabla (Triggers)**: un trigger es una propiedad de una tabla. Usted no puede llamar un trigger. Los triggers son llamados automáticamente por el motor de la base 4D cada vez que se efectúa una operación en los registros de una tabla (adición, eliminación y modificación). Los triggers son métodos que pueden evitar operaciones "ilegales" en registros de su base. Por ejemplo, en un sistema de facturación, puede evitar que un usuario añada facturas sin especificar el cliente. Los triggers son una herramienta poderosa para controlar las operaciones en una tabla, como también para evitar pérdidas accidentales de datos o vulneración de datos. Puede escribir triggers muy sencillos y volverlos cada vez más sofisticados.

Para información detallada sobre Triggers, consulte la sección **Triggers**.

- **Métodos de proyecto**: a diferencia de los métodos de objeto, métodos de formulario, y triggers, los cuales están asociados con un objeto, formulario, o tabla en particular, los métodos de proyecto están disponibles para ser utilizados en su base de datos. Los métodos de proyecto son reutilizables, y están disponibles para ser utilizados por cualquier otro método. Si necesita repetir una tarea, no tiene que escribir métodos idénticos para cada caso. Puede llamar los métodos de proyecto cuando los necesite, desde otros métodos de proyecto o desde métodos de objeto o formulario. Cuando llama un método de proyecto, se comporta como si usted hubiera escrito el método en el lugar de donde lo llamó. Los métodos de proyecto utilizados por otros métodos son llamados "subrutinas." Un método de proyecto que devuelve un resultado también puede llamarse **función**.

Hay otra manera de utilizar métodos de proyecto, asociándolos a comandos de menús. Cuando asocia un método de proyecto con un comando de menú, el método se ejecuta cuando el comando de menú es seleccionado.

Para mayor información sobre métodos de proyecto, consulte la sección **Métodos de proyecto**.

- **Métodos de base de datos**: al igual que los métodos de objeto y de formulario que se llaman cuando ocurre un evento en un formulario, hay métodos asociados con la base de datos que se llaman cuando ocurre un evento de sesión de trabajo. Estos son los **métodos de bases de datos**. Por ejemplo, cada vez que abre una base de datos, podría inicializar algunas variables que se utilizarán durante toda la sesión de trabajo. Para hacerlo, utilice Método de base On Startup, que se ejecuta automáticamente por 4D cuando abre la base.

Para mayor información sobre métodos de base de datos, consulte la sección **Métodos de base de datos**.

Un ejemplo de método de proyecto

Todos los métodos son fundamentalmente iguales, comienzan en la primera línea y trabajan a través de cada línea de instrucción hasta que alcanzan la última línea (es decir, se ejecutan secuencialmente). Este es un ejemplo de método de proyecto:

```
QUERY([Personas])  ` Muestra el editor de búsquedas
If(OK=1)  ` El usuario hace clic en OK, no cancela
   If(Records in selection([Personas])=0)  ` Si no se encuentra ningún registro···
```

```
      ADD RECORD([Personas]) ` Permítale al usuario añadir un nuevo registro
   End if
End if ` Fin
```

Cada línea en el ejemplo es una **instrucción** o línea de código. Todo lo que escriba utilizando el lenguaje se llama código. Él se ejecuta; esto significa que 4D efectúa la tarea especificada por el código.

Examinaremos la primera línea en detalle y luego avanzaremos más rápidamente:

```
QUERY([Personas]) ` Mostrar el editor de búsquedas
```

El primer elemento en la línea, **QUERY**, es un comando. Un comando es una parte del lenguaje de 4D, que realiza una tarea. En este caso, **QUERY** muestra el editor de búsquedas. Esto es similar a seleccionar Buscar en el menú Registros en el entorno Diseño.

El segundo elemento en la línea, especificado entre paréntesis, es un argumento para el comando **QUERY**. Un argumento (o **Parámetro**) es un valor necesario para un comando para poder completar su tarea. En este caso, *[Personas]* es el nombre de una tabla. Los nombres de tablas siempre se especifican en corchetes (*[...]*). En nuestro ejemplo, la tabla Personas es un argumento del comando **QUERY**. Un comando puede aceptar varios parámetros.

El tercer elemento es un **comentario** al final de cada línea. Un comentario le informa a usted (y a cualquier persona que lea su código) que pasa en el código. Un comentario se señala con el apostrofe inverso (`). En una línea, todo lo que sigue después de una marca de comentario se ignora cuando el código se ejecuta. Un comentario puede ser colocado sólo en una línea, o se puede colocar comentarios a la derecha del código, como en el ejemplo. Utilice generosamente los comentarios en su código; esto hace que sea más fácil para usted y para otros leer y entender el código.

Nota: un comentario puede tener hasta 32 000 caracteres.

La siguiente línea del método verifica si se encontró algún registro:

```
If(Records in selection([Personas])=0) ` Si no se encontró ningún registro···
```

La instrucción **If** es una **instrucción de control de flujo**, una instrucción que controla la ejecución paso a paso de su método. La instrucción **If** realiza una prueba, y si la instrucción es verdadera, se continúa con la ejecución de las líneas siguientes. **Records in selection** es una función, un comando que devuelve un valor. Acá, **Records in selection** devuelve el número de registros en la selección actual para la de la tabla pasada como argumento.

Nota: observe que sólo la primera letra del nombre de la función es mayúscula. Esta es la convención de escritura utilizada por la funciones de 4D.

Usted ya sabe lo que es una selección actual, es el grupo de registros con el cual está trabajando en un momento dado. Si el número de registros es igual a 0 (en otras palabras, si no se encuentran registros), entonces se ejecuta la siguiente línea:

```
ADD RECORD([Personas]) ` Permite que el usuario añada un nuevo registro
```

El comando **ADD RECORD** muestra un formulario de manera que el usuario pueda añadir un nuevo registro. 4D le da formato a su código automáticamente; observe que esta línea está indentada para mostrarle que está dependiendo de la instrucción de flujo de control (If).

```
End if ` Fin
```

La instrucción **End if** concluye la secuencia de instrucciones controlada por **If**. Donde haya una instrucción de flujo de control, debe tener una instrucción correspondiente indicándole al lenguaje donde detener el control.

Asegúrese de sentirse cómodo con los conceptos de esta sección. Si son todos nuevos, podría revisarlos hasta que aclararlos.

¿A dónde ir ahora?

A aprender más sobre:

- Métodos de objetos y métodos de formulario, consulte la sección **Eventos de formulario**.
- Triggers, consulte la sección **Triggers**.
- Métodos de proyecto, consulte la sección **Métodos de proyecto**.
- Métodos de base de datos, consulte la sección **Métodos de base de datos**.

Métodos de proyecto

Los métodos de proyecto, como su nombre lo indica, se aplican a la totalidad de su proyecto. Mientras los métodos de formulario y de objeto están asociados a formularios y objetos, un método de proyecto está disponible en todas partes; no está asociado específicamente a ningún objeto en particular de la base. Un método de proyecto puede tener uno de los siguientes papeles, dependiendo de cómo se ejecute y utilice:

- Método de menú
- Subrutina y función
- Método de proceso
- Método de gestión de eventos
- Método de gestión de errores

Estos términos no distinguen los métodos de proyecto por lo que son, si no por lo que hacen.

Un **método menú** es un método de proyecto llamado desde un menú personalizado, que dirige el flujo de su aplicación. Los métodos de menú controlan, cuando es necesario, la presentación de formularios, la generación de informes y en general administración de su base de datos.

La **subrutina** es un método de proyecto que puede ser como considerado como un empleado. Realiza aquellas tareas que otros métodos le piden que realice. Una función es una subrutina que devuelve un valor al método que la llamó.

Un **método de proceso** es un método de proyecto que se llama cuando se inicia un proceso. El proceso dura mientras el método de proceso se ejecuta. Para mayor información sobre procesos, consulte las sección **Procesos**. Observe que un método de menú asociado a un comando de menú para el cual la propiedad **Nuevo proceso** está seleccionada, también es el método de proceso para el nuevo proceso iniciado.

Un **método de gestión de eventos** se ejecuta en un proceso separado como el método de proceso de gestión de eventos. Generalmente, usted le permite a 4D hacer la mayoría del trabajo de gestión de eventos. Por ejemplo, durante la entrada de datos, 4D detecta los clics y las teclas presionadas, luego llama los métodos de objeto y formulario correspondientes, de manera que responda apropiadamente a los eventos desde estos métodos. En otras circunstancias, usted podría querer manejar directamente los eventos. Por ejemplo, si ejecuta una operación larga (tal como un bucle **For...End for** a cada registro), usted podría interrumpir la operación al presionar Ctrl-Punto (Windows) o Cmd-Punto (Macintosh). En este caso, debe utilizar un método de gestión de eventos para hacerlo. Para mayor información, consulte la descripción del comando **ON EVENT CALL**.

Un **método de gestión de errores** es un método de proyecto de interrupción. Cada vez que un error o una excepción ocurre, se ejecuta en el proceso en el cual fue instalado. Para mayor información, vea la descripción del comando **ON ERR CALL**.

Métodos de menú

Un método de menú se llama en el entorno de Aplicación cuando se selecciona el comando de menú al que está asociado. Usted asigna el método al comando de menú en el editor de menús. El menú se ejecuta cuando el comando de menú es seleccionado. Este proceso es uno de los principales aspectos de la personalización de una base de datos. Creando menús personalizados con métodos de menú que realizan acciones específicas, usted personaliza su base. Consulte el Manual de Diseño para mayor información sobre el editor de menús.

Los comandos de menús personalizados pueden hacer que una o más actividades tomen lugar. Por ejemplo, un comando de menú para introducir registros podría llamar un método que efectúe dos tareas: mostrar el formulario de entrada apropiado, y llamar al comando **ADD RECORD** hasta que el usuario cancele la entrada de datos.

La automatización de secuencias de acciones es una posibilidad muy poderosa del lenguaje de programación. Utilizando menús personalizados, puede automatizar secuencias de tareas y ofrecer mayor orientación a los usuarios de la base de datos.

Subrutinas

Cuando crea un método de proyecto, el método se vuelve parte del lenguaje de la base en la cual fue creado. Entonces puede llamar al método de proyecto de la misma forma como llama a los comandos integrados de 4D. Un método de proyecto utilizado de esta manera se llama una subrutina.

Usted utiliza subrutinas para:

- Reducir código repetitivo
- Aclarar sus métodos
- Facilitar los cambios en sus métodos
- Modularizar su código

Por ejemplo, imagine que tiene una base de clientes. A medida que personaliza la base, encuentra que hay algunas tareas que usted realiza repetidamente, tales como encontrar un cliente y modificar su registro. El código para hacer esto se vería así:

```
  ` Buscar un cliente
QUERY BY EXAMPLE([Clientes])
  ` Seleccionar el formulario de entrada
FORM SET INPUT([Clientes];"Entrada de datos")
  ` Modificación del registro del cliente
MODIFY RECORD([Clientes])
```

Si no utiliza subrutinas, tendrá que escribir el código cada vez que quiera modificar el registro de un cliente. Si esta operación se realiza en su base personalizada diez veces, usted tendrá que escribir el código diez veces. Si utiliza subrutinas, sólo tendrá que

escribir el código una vez. Esta es la primera ventaja de las subrutinas, reducir la cantidad de código.

Si el código descrito anteriormente fuera un método llamado **MODIFICAR CLIENTE**, lo ejecutaría simplemente utilizando el nombre del método en otro método. Por ejemplo, para modificar el registro de un cliente y luego imprimirlo, debe escribir este método:

```
MODIFICAR CLIENTE
PRINT SELECTION([Clientes])
```

Esta posibilidad simplifica dramáticamente sus métodos. En el ejemplo, usted no necesita saber cómo funciona el método **MODIFICAR CLIENTE**, sólo lo que hace. Esta es la segunda razón para utilizar subrutinas, para aclarar sus métodos. De esta forma, sus métodos se vuelven extensiones del lenguaje de 4D.

Si tiene que cambiar su método de buscar clientes en esta base de ejemplo, tendrá que cambiar sólo un método, no diez. Esta es la siguiente razón para utilizar subrutinas, facilitar los cambios en sus métodos.

Utilizando subrutinas, usted hace su código modular. Esto significa simplemente dividir su código en módulos (subrutinas), cada una de las cuales realiza una tarea lógica. Examine el siguiente código de una base de una cuenta corriente:

```
BUSCAR CHEQUES EMITIDOS  ` Buscar los cheques emitidos
CONCILIAR CUENTA  ` Conciliar la cuenta
IMPRIMIR INFORME CHEQUERA  ` Imprimir un informe de chequera
```

Incluso para alguien que no conoce la base, el código es claro. No es necesario examinar cada subrutina. Cada subrutina podría tener muchas líneas y realizar algunas operaciones complejas, pero acá lo único importante es que realice su tarea.

Recomendamos que divida su código en tareas lógicas, o módulos, cada vez que sea posible.

Pasar parámetros a los métodos

Con frecuencia encontrará que necesita pasar datos a sus métodos. Esto se hace fácilmente con parámetros.

Los Parámetros (o argumentos) son los datos que un método necesita para realizar su tarea. Los términos parámetro y argumento se utilizan indiferentemente en este manual. Igualmente, los parámetros se pasan a los comandos integrados de 4D. En este ejemplo, la cadena "Hola" es un argumento del comando **ALERT**:

```
ALERT("Hola")
```

Los parámetros se pasan a los métodos de la misma forma. Por ejemplo, si un método llamado **HACER ALGO** acepta tres parámetros, una llamada al método podría verse de esta forma:

```
HACER ALGO(ConEsto;yEsto;Asi)
```

Los parámetros se separan por punto y comas (*;*).

En la subrutina (el método llamado), el valor de cada parámetro se copia automáticamente de manera secuencial en las variables locales numeradas: *$1*, *$2*, *$3*, etc. La numeración de las variables locales representa el orden de los parámetros.

```
 //Código del método DO SOMETHING
 //Asumiendo que todos los parámetros son de tipo texto
C_TEXT($1;$2;$3)
ALERT("I received "+$1+" and "+$2+" and also "+$3)
 //$1 contiene el parámetro WithThis
 //$2 contiene el parámetro AndThat
 //$3 contiene el parámetro ThisWay
```

Dentro de la subrutina, puede utilizar los parámetros *$1*, *$2...* de la misma forma que utilizaría cualquier otra variable local.

Sin embargo, en caso de que utilice los comandos que modifican el valor de la variable pasada como parámetro (por ejemplo, **Find in field**), los parámetros *$1*, *$2, etc.* no pueden utilizarse directamente. Primero debe copiarlos en las variables estándar locales (por ejemplo: *$mivar:=$1*).

Nota avanzada: los métodos de proyecto 4D aceptan un número variable de parámetros del mismo tipo, empezando por la derecha. Para declarar estos parámetros, se utiliza una directiva de compilador a la que se lepasa ${N} como parámetro, donde N especifica el primer parámetro. Por ejemplo, la declaración **C_LONGINT**(${5}) indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de tipo entero largo. Utilizando el comando [#cmd id="259"/] puede entonces tratar esos parámetros con un bucle For y la sintaxis de indirección de parámetros. Para más información, consulte el ejemplo 2 del comando **Count parameters**.

of the local parameter will modify the source value.Dependiendo de su tipo, los parámetros se pasan **por copia** o **por referencia**:

- Cuando un parámetro se pasa **por copia**, las variables/parámetros locales no son los campos, las variables o las expresiones reales pasadas por el **método llamante**; Sólo contienen los valores que se han pasado. Dado que su alcance es local, si el valor de un parámetro se modifica en la subrutina, no cambia el valor en el método de llamada
- Cuando un parámetro se pasa **por referencia**, las variables/parámetros locales contienen referencias que apuntan a los campos fuente, variables o expresiones reales pasadas por el **método llamante**; Modificar el valor del parámetro local modificará el valor de la fuente.

La siguiente tabla muestra cómo se pueden pasar los diferentes tipos de elementos:

Tipo de parámetro	Cómo se pasa	Comentario
Campo, variable o expresión de un tipo escalar (número, texto, fecha...)	por valor	Se puede pasar por referencia a través de un puntero, ver abajo
Campo, variable o expresión de tipo Objeto	por referencia	Ver ejemplo a continuación
Variable o expresión de tipo Colección	por referencia	
Variable o expresión de tipo Puntero	por referencia	Ver **Pasar punteros a métodos**
Array	No se puede pasar directamente como parámetro	Se puede pasar por referencia a través de un puntero, ver **Arrays y punteros**
Table	No se puede pasar directamente como parámetro	Se puede pasar por referencia a través de un puntero, ver **Punteros**

Parámetros pasados por valor

Cuando se utilizan campos, variables y expresiones de tipo escalar como parámetros de método, sólo se pasan copias de valores. Como *$1*, *$2...* son variables locales, están disponibles sólo dentro de la subrutina y son borradas al final de la subrutina. Por esta razón, una subrutina no puede cambiar el valor real de los campos o variables pasadas como parámetros al nivel del método llamante. Por ejemplo:

```
` He aquí parte del método MI MÉTODO
` ...
HACER ALGO([Personas]Apellido) ` Supongamos que [Personas]Apellido es igual a "pérez"
ALERT(([Personas]Apellido)

` Este es el código del método HACER ALGO
$1:=Uppercase($1)
ALERT($1)
```

La caja de diálogo de alerta mostrada por ***HACER ALGO*** contendrá "PÉREZ" y la caja de diálogo mostrada por ***MI METODO*** contendrá "pérez". El método ha modificado localmente el valor del parámetro *$1*, pero esto no afecta el valor del campo *[Personas]Apellido* pasado como parámetro por el método ***MI METODO***.
Hay dos maneras de hacer que el método ***HACER ALGO*** cambie el valor del campo:

1. En lugar de pasar el campo al método, pasar un apuntador, escriba:

```
` He aquí parte del método MI METODO
` ...
HACER ALGO(->[Personas]Apellido) ` Digamos que [Personas]Apellido es igual a "pérez"
ALERT([Personas]Apellido)

` Este es el código del método HACER ALGO
$1->:=Uppercase($1->)
ALERT($1->)
```

Aquí el parámetro no es el campo, sino un puntero al campo. Por lo tanto, dentro del método ***HACER ALGO***, *$1* ya no es el valor del campo sino un puntero hacia el campo. El objeto referenciado por *$1 ($1->* en el código anterior) es el campo real. Por consiguiente, la modificación del objeto referenciado va más allá del alcance de la subrutina, y el campo real es afectado. En este ejemplo, ambas cajas de alerta mostrarán "PÉREZ".

Para mayor información sobre **Punteros**, consulte la sección **Punteros**.

2. En lugar de tener al método ***HACER ALGO*** "haciendo algo", puede reescribir el método de manera que devuelva un valor. De esta forma usted escribiría:

```
` He aquí parte del código del método MI METODO
` ...
[Personas]Apellido:=HACER ALGO([Personas]Apellido) ` Supongamos que [Personas]Apellido es igual a "pérez"
ALERT([Personas]Apellido)
` Here the code of the method <span class="rte4d_met">HACER ALGO</span>
$0:=Uppercase($1)
ALERT($0)
```

Esta segunda técnica de que una subrutina devuelva un valor se llama "utilizar una función." Este punto se describe en los párrafos siguientes.

Parámetros pasados por referencia

Cuando se utilizan variables, expresiones o campos de tipo objeto o colección como parámetros de método, se pasan referencias a los valores fuente reales. En este caso, *$1, $2...* no contienen valores sino referencias. La modificación del valor de los parámetros *$1*, *$2*... dentro de la subrutina se propagará donde quiera que se utilice el objeto o la colección fuente. Este es el mismo principio que para los punteros, excepto que los parámetros *$1, $2...* no necesitan ser desreferenciados en la subrutina.

Por ejemplo:

```
//El método CreatePerson crea un objeto y lo envía como un parámetro
C_OBJECT($person)
```

```
$person:=New object("Name";"Smith";"Age";40)
ChangeAge($person)
ALERT(String(OB get($person;"Age")))
```

```
 //El método ChangeAge añade 10 al atributo Age del objeto recibido
C_OBJECT($1)
OB SET($1;"Age";OB Get($1;"Age")+10)
ALERT(String(OB get($1;"Age")))
```

Si ejecuta el método ***CreatePerson***,
ambos cuadros de alerta dirán "50" ya que la misma referencia es manejada por ambos métodos.

4D Server: cuando los parámetros se pasan entre métodos que no se ejecutan en la misma máquina (utilizando por ejemplo la opción **Ejecutar en servidor**, ver **Propiedades de los métodos proyecto**), las referencias no se pueden utilizar. En estos casos, se envían copias de los parámetros de objeto y colección en lugar de referencias.

Funciones: métodos de proyecto que devuelven un valor

Los métodos pueden devolver valores. Un método que devuelve un valor se llama **función**.

Los comandos de 4D o de los plug-ins que devuelven un valor también se llaman funciones.

Por ejemplo, la siguiente línea es una instrucción que utiliza la función integrada, **Length**, para devolver la longitud de una cadena. La instrucción coloca el valor devuelto por **Length** en una variable llamadaMiLongitud. Esta es la instrucción:

```
MiLongitud:=Length("¿'Cómo llegué acá?")
```

Toda subrutina puede devolver un valor. El valor a devolver se coloca en la variable local *$0*.

Por ejemplo, la siguiente función, llamada **Mayusculas4**, devuelve una cadena con los primeros cuatro caracteres de la cadena pasados a mayúsculas:

```
$0:=Uppercase(Substring($1;1;4))+Substring($1;5)
```

El siguiente es un ejemplo que utiliza la función **Mayusculas4** :

```
NuevaFrase:=Mayusculas4("Esto está bien.")
```

En este ejemplo, la variable **NuevaFrase** toma el valor "ESTO está bien."

El **resultado de la función**, *$0*, es una variable local dentro de la subrutina. Puede ser utilizada como tal dentro de la subrutina. Por ejemplo, en el ejemplo anterior el método **HACER ALGO**, *$0* primero fue asignado al valor *$1*, luego fue utilizado como parámetro del comando **ALERT**. Dentro de la subrutina, puede utilizar *$0* de la misma forma que utilizaría otra variable local. 4D devuelve el valor de *$0* (el valor actual cuando la subrutina termina) al método llamado.

Métodos de proyecto recursivos

Los métodos de proyecto pueden llamarse entre sí. Por ejemplo:

- El método A podría llamar al método B el cual podría llamar a A, entonces A llamará a B de nuevo, etc.
- Un método puede llamarse a sí mismo.

Esto se llama **recursividad**. El lenguaje de 4D soporta totalmente la recursividad.

Este es un ejemplo. Supongamos que tiene una tabla *[Amigos y familiares]* compuesta del siguiente conjunto de campos:
- *[Amigos y familiares]Nombre*
- *[Amigos y familiares]NinosNombre*

Para este ejemplo, asumimos que los valores en los campos son únicos (no hay dos personas con el mismo nombre). A partir de un nombre, usted quiere construir una frase "Un amigo mio, Juan el hijo de Pedro hijo de Ana hija de Roberto hijo de Eleonor, ¡hace esto para vivir!":

1. Puede construir la frase de esta forma:

```
$vsNombre:=Request("Introduzca el nombre:";"Juan")
If(OK=1)
   QUERY([Amigos y familiares];[Amigos y familiares]Nombre=$vsNombre)
   If(Records in selection([Amigos y familiares])>0)
      $vtTodalahistoria:="Un amigo mio, "+$vsNombre
      Repeat
         QUERY([Amigos y familiares];[Amigos y familiares]NinosNombre=$vsNombre)
         $vlQueryResult:=Records in selection([Amigos y familiares])
         If($vlQueryResult>0)
            $vtTodalahistoria:=$vtTodalahistoria+" el hijo de "+[Amigos y familiares]Nombre
            $vsNombre:=[Amigos y familiares]Nombre
         End if
      Until($vlQueryResult=0)
      $vtTodalahistoria:=$vtTodalahistoria+", ¡hace esto para vivir!"
      ALERT($vtTodalahistoria)
```

```
    End if
End if
```

2. También la puede construir de esta forma:

```
$vsNombre:=Request("Introduzca el nombre:";"Juan")
If(OK=1)
    QUERY([Amigos y familiares];[Amigos y familiares]Nombre=$vsNombre)
    If(Records in selection([Amigos y familiares])>0)
        ALERT("Un amigo mio, "+Genealogia de($vsName)+", ¡hace esto para vivir!")
    End if
End if
```

con la función recursiva **Genealogia de** listada aquí:

```
` Método de proyecto Genealogia de
` Genealogia de ( Cadena ) -> Texto
` Genealogia de ( Nombre ) -> Parte de la frase

$0:=$1
QUERY([Amigos y familiares];[Amigos y familiares]NinosNombre=$1)
If(Records in selection([Amigos y familiares])>0)
    $0:=$0+" hijo de "+Genealogia de([Amigos y familiares]Nombre)
End if
```

Observe que el método **Genealogia de** se llama a sí mismo.

La primera forma es un **algoritmo iterativo**. La segunda es un **algoritmo recursivo**.

Cuando implemente código para casos como el del ejemplo anterior, es importante tener en cuenta que usted siempre puede escribir métodos utilizando iteración y recursividad. Generalmente, la recursividad ofrece código más conciso, fácil de leer y de mantener, pero su uso no es obligatorio.

Algunos usos típicos de recursividad en 4D son:

- Tratar registros dentro de tablas que relacionan unas a otras de la misma manera que en el ejemplo.
- Navegar en los documentos y carpetas de su disco, utilizando los comandos **FOLDER LIST** y **DOCUMENT LIST**. Una carpeta puede contener carpetas y documentos, las subcarpetas pueden contener carpetas y documentos, y así sucesivamente.

Importante: las llamadas recursivas deben terminar en algún momento. En el ejemplo, el método **Genealogia de** deja de llamarse a sí mismo cuando la búsqueda no devuelve registros. Sin esta condición, el método se llamaría a sí mismo indefinidamente; finalmente, 4D devolvería un error "Pila llena" porque no habría espacio para "apilar" las llamadas (así como los parámetros y las variables locales utilizadas en el método).

Depurador

- ¿Por qué un depurador?
- Ventana de error sintáctico
- Depurador
- Panel de expresión
- Panel del canal de llamada
- Panel de evaluación
- Panel de evaluación de métodos
- Puntos de interrupción
- Lista de puntos de interrupción
- Capturas de comandos
- Atajos del depurador

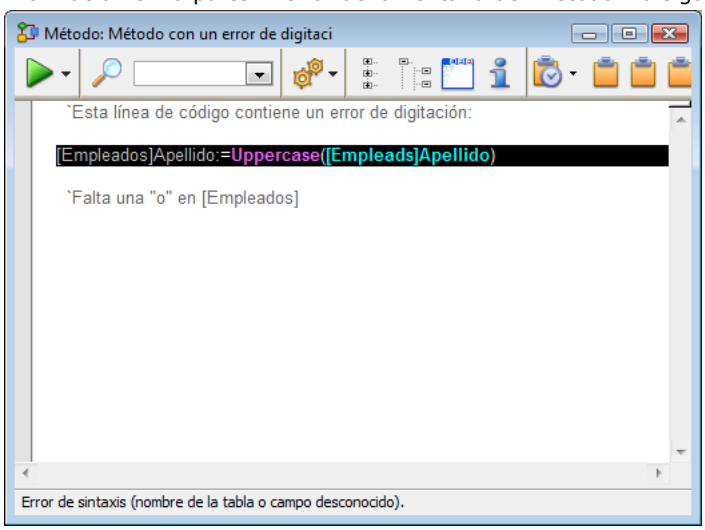
¿Por qué un depurador?

Cuando desarrolla y prueba sus métodos, es importante encontrar y arreglar los errores que puedan tener.

Hay varios tipos de errores que puede cometer cuando utiliza el lenguaje: errores de digitación, errores de sintaxis o errores de entorno, errores de diseño o lógicos y errores de ejecución (Runtime).

Errores de digitación

Los errores de digitación son detectados por el **Editor de métodos,** se muestran en rojo y aparece un mensaje en el área de información en la parte inferior de la ventana del método. La siguiente ventana muestra un error de digitación:



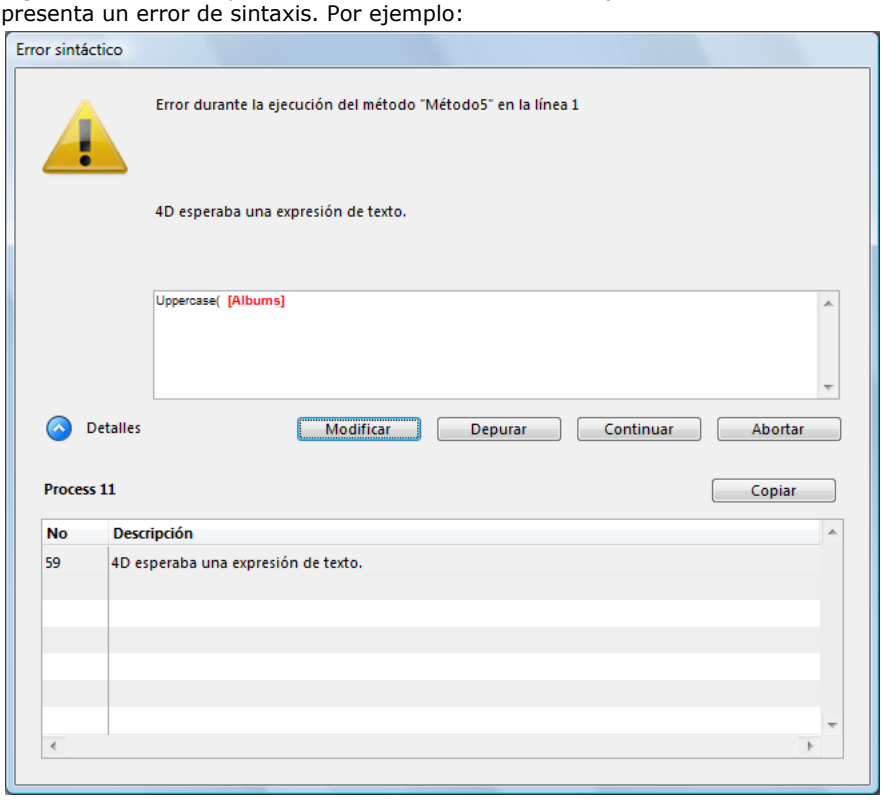
Nota: los comentarios han sido insertados manualmente para este manual. Sólo el color es modificado por 4D con respecto al error.

Generalmente tales errores de digitación producen errores de sintaxis (en este caso, el nombre de la tabla es desconocido). La barra de estado muestra una descripción del error en el momento de la validación de la línea de código.

Cuando esto ocurra, corrija el error de digitación y presione la tecla Intro (en el teclado numérico) para validar la corrección. Para mayor información sobre el editor de métodos, consulte el Manual de Diseño.

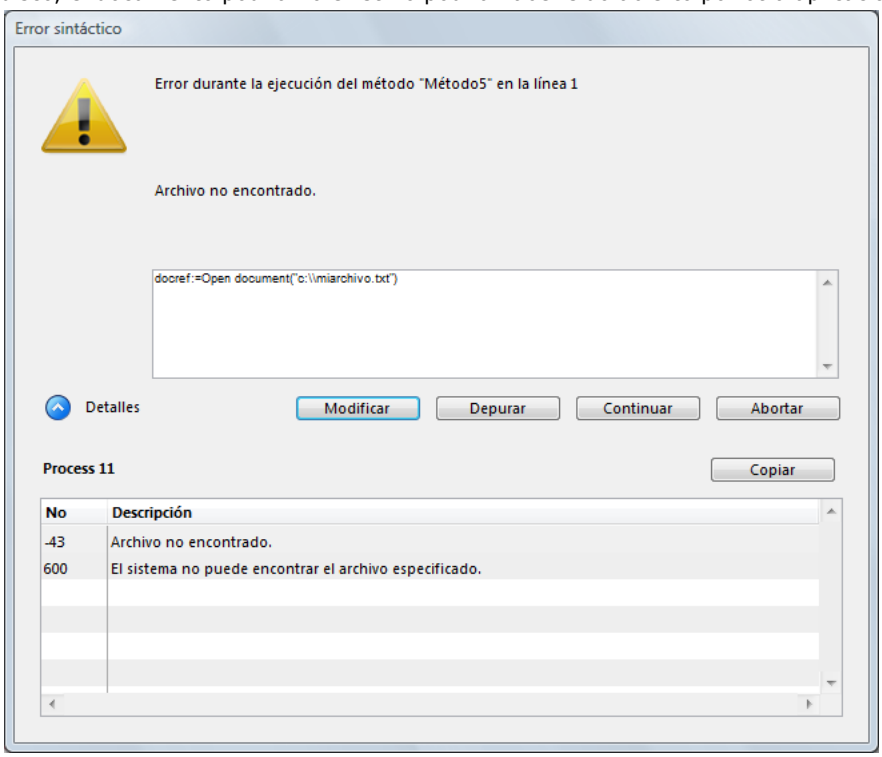
Errores de sintaxis o errores de entorno

Algunos errores sólo pueden ser detectados cuando se ejecuta el método. La *ventana Error sintáctico* aparece cuando se presenta un error de sintaxis. Por ejemplo:



En esta ventana, el error es que se pasa un nombre de tabla al comando **Uppercase**, el cual espera una expresión de texto. Para aprender más sobre esta ventana y su botón, consulte la sección **Ventana de error sintáctico**. En la imagen anterior, el área "Detalles" se expande para mostrar el último error y su número.

Ocasionalmente, podría no haber suficiente memoria para crear un Array o un BLOB. Cuando usted accede a un documento en disco, el documento podría no existir o podría haber sido abierto por otra aplicación.



Estos errores no ocurren directamente por su código o la forma en que lo escribió; simplemente ocurren. La mayoría de las veces, estos errores son fáciles de identificar con un método de interceptación de errores instalado utilizando el comando **ON ERR CALL**. Consulte la descripción de **ON ERR CALL**.

Para mayor información sobre esta ventana, consulte la sección **Ventana de error sintáctico**.

Errores de diseño o lógica

Estos errores son generalmente los más difíciles de encontrar, utilice el **Depurador** para detectarlos. Note que a excepción de los errores de digitación, todos los tipos de errores anteriores están hasta cierto punto cubiertos por la expresión "Error de diseño o lógica." Por ejemplo:

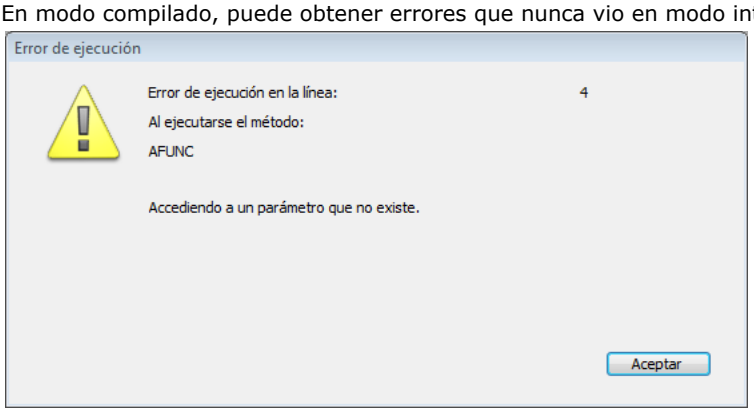
- Un error de sintaxis podría ocurrir porque usted trata de utilizar una variable que aún no ha sido inicializada.
- Un error de entorno podría producirse porque usted trata de abrir un documento cuyo nombre es recibido por una subrutina que no recibe el valor correcto en el parámetro. Note que en este ejemplo, la parte del código que no funciona podría ser diferente al código que realmente es el origen del problema.

Los errores de lógica o de diseño también incluyen situaciones como que:

- Un registro no se actualiza correctamente porque, al llamar a **SAVE RECORD**, usted olvidó probar primero si el registro estaba bloqueado o no.
- Un método no hace exáctametne lo que usted espera, porque no se prueba la presencia de un parámetro opcional.

Errores de ejecución

En modo compilado, puede obtener errores que nunca vio en modo interpretado. Este es un ejemplo:



Este mensaje indica que usted está tratando de acceder a un carácter cuya posición se encuentra más allá de la longitud de la cadena. Para encontrar rápidamente el origen del problema, anote el nombre de método y el número de la línea, abra nuevamente su base en modo interpretado y vaya al método y línea indicados.

¿Qué hacer cuando ocurre un error?

Los errores son comunes. Sería inusual escribir una gran cantidad de líneas de código sin que se generen errores. Igualmente, tratar y/o corregir errores ¡también es normal!

Gracias a su entorno multitareas, 4D le permite modificar y ejecutar rápidamente sus métodos simplemente pasando de una ventana a otra. Descubrirá lo rápido que puede arreglar los errores cuando no tiene que ejecutarlo todo nuevamente cada vez. También descubrirá cómo puede encontar errores rápidamente utilizando el **Depurador**.

Un error común que cometen los principiantes cuando detectan un error es hacer clic en Abortar en la ventana Error sintáctico, regresar al editor de métodos y tratar de encontrar el problema mirando el código. ¡No haga eso! Ahorrará mucho tiempo y energía utilizando **siempre** el **Depurador**.

- Si ocurre un error de sintaxis inesperado, utilice el **Depurador**.
- Si ocurre un error de entorno, utilice el **Depurador**.
- Si se produce otro tipo de error, utilice el **Depurador**.

En el 99% de los casos, el **Depurador** muestra la información que usted necesita para entender por qué se presentó el error. Una vez tenga esta información, sabrá cómo arreglar el error.

Consejo: pasar algunas horas aprendiendo y experimentando con el **Depurador** puede ahorrarle días y semanas en el futuro cuando tenga que encontrar errores.

Otra razón para utilizar el **Depurador** es para desarrollar código. Algunas veces usted podría escribir un algoritmo que es más complejo de lo habitual. Una vez escrito el código, usted no está totalmente seguro de que su código es correcto, incluso antes de probarlo. En lugar de ejecutarlo a "ciegas," utilice el comando **TRACE** comando al comienzo de su código. Luego, ejecute paso a paso para controlar lo que sucede y verificar si sus sospechas eran ciertas o no. A un purista puede no gustarle este método, pero algunas veces el pragmatismo da resultados más rápidamente. De todas formas ... utilice el **Depurador**.

Conclusión general

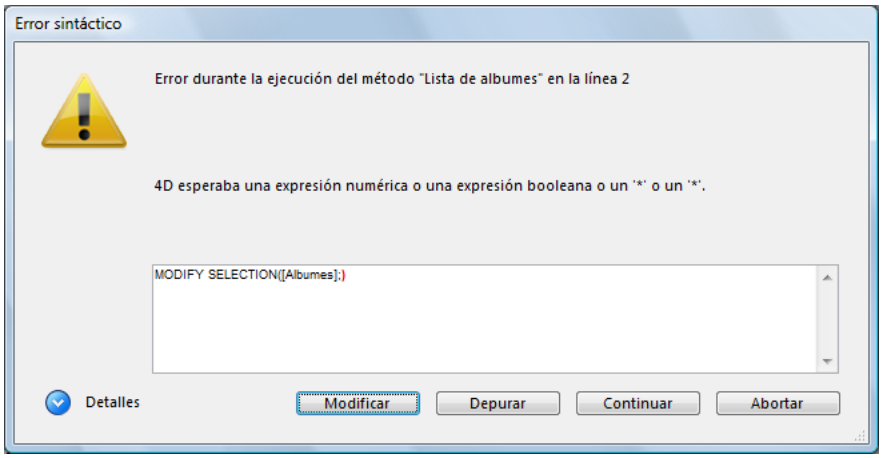
Utilice el **Depurador**.

Ventana de error sintáctico

La **Ventana de error sintáctico** se muestra cuando se interrumpe la ejecución de un método. La ejecución del método puede interrumpirse por una de las siguientes razones:

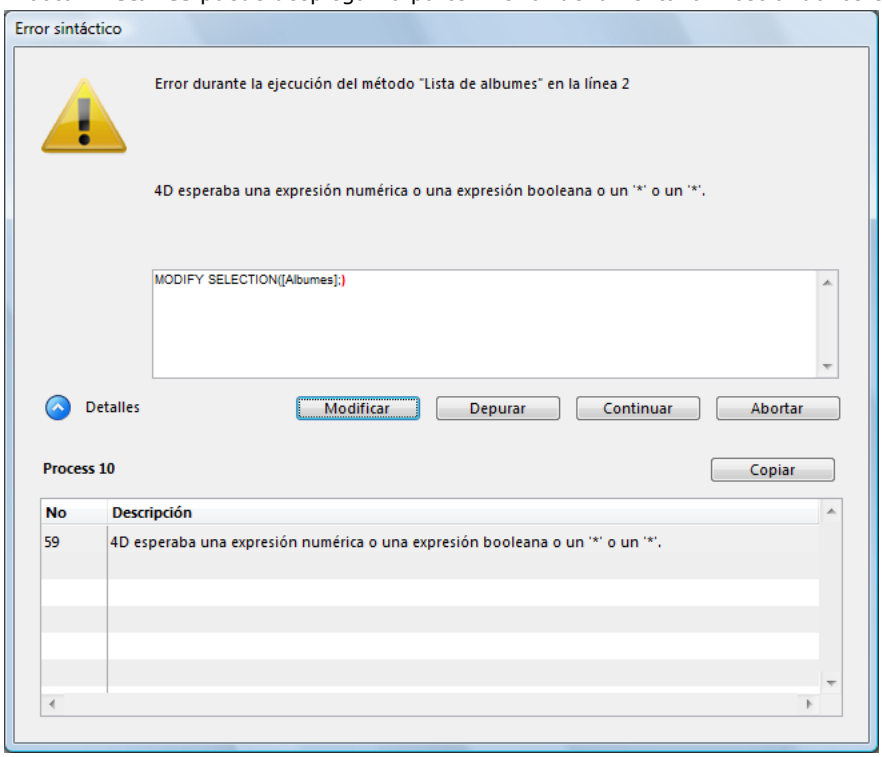
- 4D interrumpe el método porque un error le impide continuar con su ejecución.
- El método produce una falsa afirmación (ver el comando **ASSERT**)

Esta es una **Ventana de error sintáctico**:



El texto en la parte superior de la *ventana Error sintáctico* muestra un mensaje que describe el error. La parte inferior muestra la línea que se estaba ejecutando cuando ocurrió el error.

El botón **Detalles** puede desplegar la parte inferior de la ventana mostrando los errores relacionados con el proceso:



Hay cinco botones de opción en la parte inferior de la ventana: **Abortar**, **Depurar**, **Continuar**, **Modificar** y (si la ventana está expandida) **Copiar**.

● **Abortar**: el método se interrumpe y usted vuelve a donde estaba antes de iniciar la ejecución del método. Si un método de formulario o método de objeto se ejecutan en respuesta a un evento, se detiene y usted vuelve al formulario. Si el método se ejecuta desde el entorno Aplicación, usted vuelve a este entorno.

● **Depurar**: usted entra en el modo Depurar y se muestra la ventana del **Depurador**. Si la línea actual ha sido ejecutada parcialmente, podría tener que hacer clic en el botón Depurar varias veces. Una vez la línea termina, se muestra la ventana del **Depurador**.

● **Continuar**: continúa la ejecución. La línea que contiene el error puede ejecutarse parcialmente, dependiendo de dónde se encuentre el error. Continúe con prudencia, el error podría evitar que el resto del método se ejecute correctamente. Generalmente no querrá continuar. Puede hacer clic en Continuar si el error está en una llamada trivial, tal como **SET WINDOW TITLE**, que no impide ejecutar y probar el resto de su código. Entonces puede concentrarse en el código más importante y corregir los errores menores más adelante.

Nota: si presiona la tecla **Alt** (Windows) u **Opción** (Mac OS), el botón **Continuar** cambia a **Ignorar**. Hacer clic en **Ignorar**

significa que la ventana no se visualizará si el mismo error, provocado por el mismo método en la misma línea, se produce de nuevo. Este acceso directo es útil en el caso de un error que se produce repetidamente, por ejemplo en un bucle. En este caso, todo sigue como si el usuario hiciera clic en el botón **Continuar** cada vez.

● **Modificar**: la ejecución del método se interrumpe totalmente. 4D pasa al entorno Diseño. El método en el cual ocurrió el error se abre en el editor de métodos, permitiéndole corregir el error. Utilice esta opción cuando haya identificado el error y pueda corregirlo sin necesidad de realizar más investigaciones.

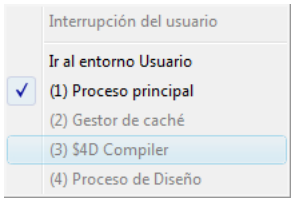
● **Copiar**: este botón copia la información del depurador en el portapapeles. Esta información describe el entorno interno del error (número, componente interno, etc.). Es formateado como texto tabulado. Una vez haga clic en este botón, puede pegar el contenido del portapapeles en un archivo de texto, hoja de cálculo, un e-mail, etc. por propósitos de análisis.

Depurador

La palabra Depurador viene del término inglés bug (insecto). Un bug en un método es un error que usted quiere eliminar. Cuando ocurra un error, o cuando necesite monitorear la ejecución de sus métodos, utilice el depurador. Un depurador le ayuda a encontrar bugs al permitirle ir lentamente paso a paso a través de sus métodos y examinar la información del método. Este proceso se llama **depuración**.

Puede hacer que la ventana del Depurador aparezca y luego depurar los métodos de estas formas:

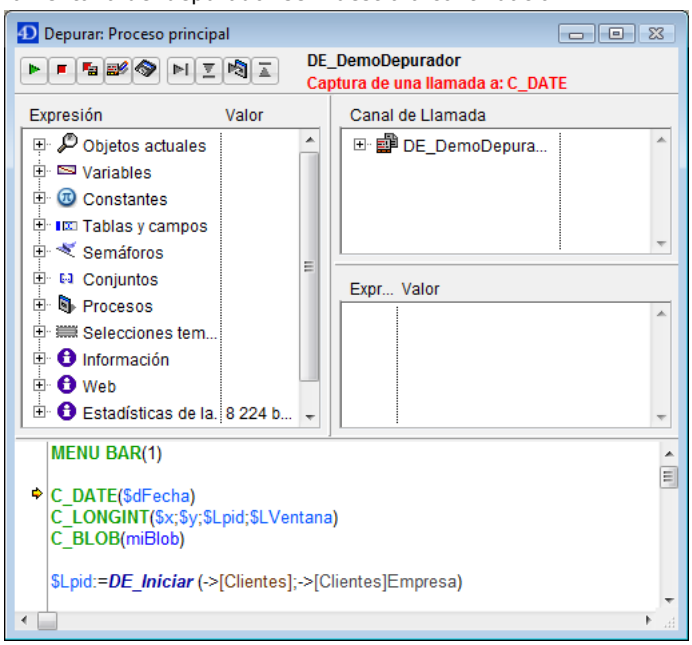
- Haciendo clic en el botón **Depurar** en la ventana de error de sintaxis
- Utilizando el comando **TRACE**
- Haciendo clic en el botón **Depurar** en la ventana de ejecución del método (Entorno usuario).
- Presionando **Alt+Mayúsc+Clic derecho** (Windows) o **Control+Opción+Comando+Clic** (Macintosh) mientras se ejecuta un método, luego seleccione el proceso a depurar en el menú desplegable:



- Haciendo clic en el botón **Depurar** cuando se selecciona un proceso en la página **Proceso** del **Explorador de ejecución**.
- Creando o editando un punto de interrupción en la ventana del editor de métodos, o en las páginas **Punto de interrupción** y **Captura** del **Explorador de ejecución**.

Nota: si existe un sistema de contraseñas para la base, sólo el diseñador y los usuarios que pertenezcan al grupo que tiene privilegios de acceso a la estructura pueden depurar métodos.

La ventana del depurador se muestra a continuación:

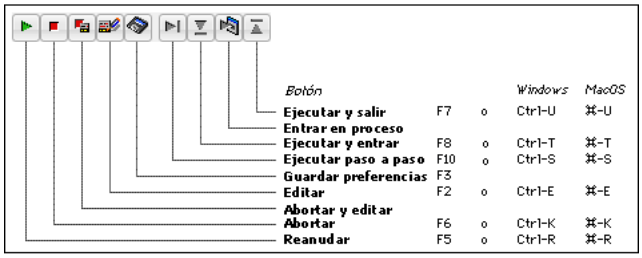


Puede mover la ventana del depurador y/o redimensionar sus paneles internos según sea necesario. La visualización de una nueva ventana del depurador utiliza la misma configuración (tamaño y posición de la ventana, ubicación de las líneas de división y contenido del área de evaluación de expresiones) de la ultima ventana visualizada en la misma sesión.

4D es un entorno multitareas. Si ejecuta varios procesos simultáneamente, puede depurarlos independientemente. Es posible tener una ventana de depurador abierta para cada proceso.

Botones de la barra de herramientas de control de ejecución

La **Barra de herramientas de control de ejecución** situada en la parte superior de la ventana del Depurador, tiene nueve botones:



Botón Reanudar

Se detiene la depuración y se retoma la ejecución normal del método.

Nota: **Mayús+F5** o **Mayús+clic** en el botón **Reanudar** reanuda la ejecución y desactiva todas las llamadas siguientes a **TRACE** en el proceso actual.

Botón Abortar

El método se detiene y usted vuelve a donde estaba antes de comenzar la ejecución del método. Si está depurando un método de objeto o de formulario en respuesta a un evento, se detiene y usted vuelve al formulario. Si está depurando un método ejecutado desde el entorno Menús Personalizados, usted vuelve al entorno Menús personalizados.

Botón Abortar y Editar

El método se detiene como si usted hiciera clic en Abortar. Igualmente, 4D abre, si es necesario, el proceso del entorno Diseño y lo pasa al primer plano, luego abre la ventana del editor de métodos para el método que se estaba ejecutando en el momento en que se hizo clic en el botón **Abortar y Editar**.

Consejo: utilice este botón cuando sepa cuales son las modificaciones requeridas en su código y cuando estos cambios sean requeridos para probar sus métodos. Una vez terminadas las modificaciones, ejecute nuevamente el método.

Botón Editar

Este botón se comporta como el botón **Abortar y Editar**, con la diferencia de que no aborta la ejecución actual. La ejecución del método simplemente se suspende en ese punto. Si es necesario, 4D abre el proceso del entorno Diseño y lo pasa al primer plano, luego abre una ventana del editor de métodos para el método que se estaba ejecutando en el momento en que se hizo clic en el botón **Editar**.

Importante: puede modificar este método; sin embargo, estas modificaciones no aparecerán o ejecutarán en la instancia del método que está siendo depurado actualmente en la ventana del depurador. Después de que el método haya sido abortado o completado con éxito, aparecerán las modificaciones en la siguiente ejecución de método. En otras palabras, el método debe recargarse para que las modificaciones se tengan en cuenta.

Consejo: utilice este botón cuando sepa cuales son los cambios necesarios en su código y cuando estos no interfieran con el resto del código a ejecutar o depurar.

Consejo: los métodos de objeto son recargados para cada evento. Si está depurando un método de objeto (por ejemplo, en respuesta al clic de un botón), no necesita salir el formulario. Puede modificar el método de objeto, guardar los cambios, luego volver al formulario y tratar de nuevo. Para depurar/cambiar métodos de formulario, debe salir del formulario y abrirlo nuevamente para cargar nuevamente el método de formulario. Cuando efectúa la depuración completa de un formulario, un truco es colocar el código (que usted está depurando) en un método de proyecto que usted utilizará como subrutina al interior del método del formulario. De esta forma, puede permanecer en el formulario mientras depura, edita, y prueba nuevamente su formulario, porque la subrutina se cargará cada vez que sea llamada por el método de formulario.

Botón Guardar Preferencias

Este botón guarda la configuración actual de la ventana del depurador (tamaño y posición de la ventana, ubicación de las líneas de división y contenido del área de evaluación de las expresiones), de manera que la configuración se utilice por defecto cada vez que se abra la base. Estos parámetros se almacenan en el archivo de estructura de la base.

Botón Ejecutar paso a paso

La línea actual del método (indicada por la flecha amarilla, llamada el **contador del programa**) se ejecuta y el depurador pasa a la línea siguiente. El botón **Ejecutar paso a paso** no pasa por las subrutinas y funciones; se queda en el nivel del método que usted esta depurando. Si quiere depurar también las llamadas a las subrutinas y funciones, utilice el botón **Ejecutar y Entrar**.

Botón Ejecutar y Entrar

Durante la ejecución de una línea que llama a otro método (subrutina o función), este botón provoca la visualización del método llamado en la ventana del depurador y le permite pasar paso a paso por este método. El nuevo método se convierte en el método actual (arriba) en el *panel Canal de llamada* de la ventana del depurador. En el momento de la ejecución de una línea que no llama a otro método, este botón actúa de la misma forma que el botón **Ejecutar paso a paso**.

Botón Entrar en proceso

Durante la ejecución de una línea que crea un nuevo proceso (por ejemplo, llamando al comando **New process**), este botón abre una nueva ventana de depurador que le permite depurar el método de proceso proceso creado recientemente. Durante la ejecución de una línea que no crea un nuevo proceso, este botón actúa de la misma forma que el botón **Ejecutar paso a paso**.

Botón Ejecutar y Salir

Si está depurando subrutinas y funciones, hacer clic en este botón le permitir ejecutar la integridad del método que está siendo depurado actualmente y volver al método que las llama. La ventana del depurador regresa al método anterior en la cadena de llamado. Si el método actual es el último método en la cadena de llamado, la ventana del depurador se cierra.

Información en la barra de herramientas del control de ejecución

A la derecha de la barra de herramientas de control de ejecución, el depurador suministra la siguiente información:

- El nombre del método que está depurando actualmente (en negro)

- El problema que hizo que apareciera la ventana del depurador (en rojo)

Utilizando la ventana de ejemplo mostrada anteriormente, aparece la siguiente información:

- El método **DE_DemoDepurador** es el método que está siendo depurado.
- La ventana del depurador apareció porque se detecto una llamada al comando C_DATE y este comando era uno de los comandos a ser capturados.

Estas son las posibles razones para que aparezca el depurador y un mensaje (en rojo):

- **Comando TRACE**: se ha llamado a **TRACE**.
- **Punto de interrupción alcanzado**: se ha encontrado un punto de interrupción temporal o permanente.
- **Interrupción del usuario**: usted presionó Alt+Mayús+clic derecho (Windows) o Control+Opción+comando+clic (Macintosh), o hizo clic en el botón **Depurar** en la página **Proceso** del explorador de ejecución.
- **Captura de una llamada a: Nombre del comando**: una llamada a un comando 4D que debe ser interceptado está a punto de ser ejecutada.
- **Paso a paso en nuevo proceso**: usted utilizó el botón **Entrar en proceso** y este mensaje aparece en la ventana del Depurador abierta para el proceso que acaba de ser creado.

Los paneles de la ventana del depurador

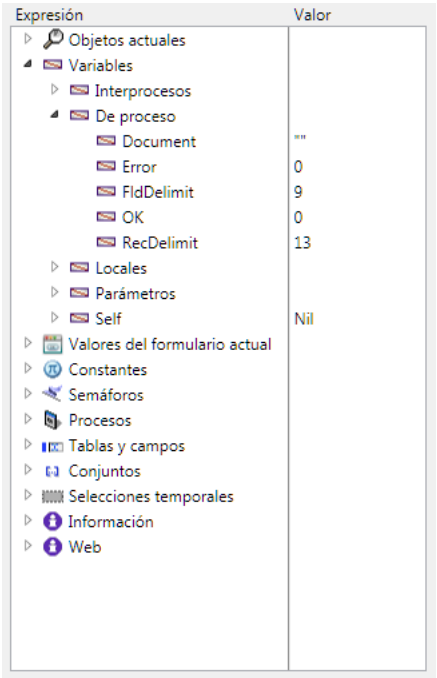
La ventana del depurador contiene la barra de control de ejecución descrita anteriormente, así como cuatro paneles redimensionables:

- **Panel de expresión**
- **Panel del canal de llamada**
- **Panel de evaluación**
- **Panel de evaluación de métodos**

Los tres primeros paneles utilizan listas jerárquicas de fácil navegación para mostrar la información de depuración pertinente. El cuarto, , muestra el código fuente del método que está siendo depurado. Cada panel tiene su propia función para asistirlo en el proceso de depuración. Puede utilizar el ratón para redimensionar vertical y horizontalmente la ventana del depurador, así como cada panel. Además, los primeros tres paneles están separados por una línea. Utilizando el ratón, puede mover esta línea para redimensionar los paneles.

Panel de expresión

El **Panel de expresión** se sitúa en la parte superior izquierda de la ventana del depurador, bajo la barra de herramientas de control de ejecución. Este es un ejemplo:



El Panel de expresión muestra información general útil sobre el sistema, el entorno 4D y el entorno de ejecución.

La columna **Expresión** muestra los nombres de los objetos y expresiones. La columna **Valor** muestra el valor actual correspondiente a los objetos y expresiones.

Al hacer clic sobre un valor de la columna derecha del panel puede modificar el valor del objeto, si es permitido.

La lista jerárquica multiniveles está organizada por temas en el nivel superior. Los temas son:

- Objetos actuales
- Variables
- Valores del formulario actual
- Constantes
- Tablas y campos
- Semáforos
- Procesos
- Tablas y campos
- Conjuntos
- Selecciones temporales
- Información
- Web

Dependiendo del tema, cada elemento puede tener uno o más subniveles. Haciendo clic en el nodo junto al nombre del tema se despliega o se cierra el tema. Si el tema se despliega, los elementos de ese tema quedan visibles. Si el tema tiene varios niveles de información, haga clic en el nodo de la lista junto a cada elemento para explorar toda la información que contiene.

En cualquier momento, puede arrastrar y soltar temas, sublistas de temas (si hay) y elementos de temas al .

Objetos actuales

Este tema muestra los valores de los objetos o expresiones que:

- se utilizan en la línea de código a ejecutar (la indicada por el contador del programa, la flecha amarilla en el **Panel de evaluación de métodos**), o
- se utilizan en la línea de código anterior.

Como la línea de código anterior es la que acaba de ser ejecutada, el tema Objetos actuales muestra los objetos o expresiones de la línea actual antes y después de que la ejecución de la línea. En la ejecución del siguiente método:

```
TRACE
a:=1
b:=a+1
c:=a+b
` ...
```

1. Usted entra en la ventana del depurador con el contador del programa del **Panel de evaluación de métodos** ubicado en la línea *a:=1*. En este momento el tema Objetos actuales muestra:

a: Indefinido

La variable *a* se muestra porque es utilizada en la línea a ejecutar (pero no ha sido iniciada aún).

2. Avanza una línea. El contador del programa ahora está situado en la línea *b:=a+1*. En este punto, el tema Objetos actuales muestra:

a: *1*
b: Indefinido

La variable *a* se muestra porque se utiliza en la línea que acaba de ser ejecutada y le fue asignado el valor numérico *1*. También se muestra porque se utiliza en la línea a ejecutar, como expresión a asignar a la variable *b*. La varibale *b* se muestra porque se utiliza en la línea a ejecutar (pero no se ha inicializado aún).

3. Nuevamente, avanza una línea. El contador del programa está ahora en la línea *c:=a+b*. En este punto, el tema Objetos actuales muestra:

c: Indefinido
a: *1*
b: *2*

La variable *c* se muestra porque se utiliza en la línea a ejecutar (pero no ha sido iniciada aún). Las variables *a* y *b* se muestran porque fueron utilizadas en la línea anterior y se utilizan en la línea a ejecutar, etc.

El tema Objetos actuales es una herramienta muy conveniente, cada vez que ejecute una línea, no tiene que introducir una expresión en , sólo mire los valores que aparecen en el tema Objetos actuales.

Variables

Este tema se compone de los siguientes subtemas:

- **Interprocesos**: muestra la lista de las variables interproceso que se están utilizando en ese momento. Esta lista puede estar vacía si usted no utiliza variables interproceso. Los valores de las variables interproceso pueden modificarse.
- **De proceso**: muestra la lista de las variables de proceso utilizadas por el proceso actual. Esta lista rara vez está vacía. Los valores de las variables de proceso pueden modificarse.
- **Locales**: muestra la lista de las variables locales utilizadas por el método que se está depurando (el que aparece en el del código). Esta lista puede estar vacía si no se utilizan variables locales o si no se han creado aún. Los valores de las variables locales pueden modificarse.
- **Parámetros**: muestra la lista de parámetros recibidos por el método. Esta lista puede estar vacía si no se han pasado parámetros al método que está siendo depurando (el que aparece en el). El valor de los parámetros puede modificarse.
- **Puntero self** : muestra un puntero hacia el objeto actual si está depurando un método de objeto. Este valor no puede modificarse.

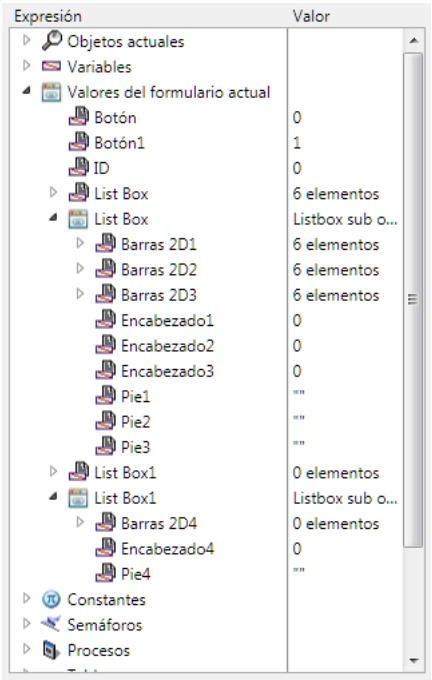
Nota: puede modificar las variables y los campos de tipo Cadena, Texto, Numérico, Fecha y Hora; en otras palabras, puede modificar las variables cuyos valores pueden ser introducidos utilizando el teclado.

Los arrays, como las otras variables, aparecen en los subtemas Interproceso, Proceso y Locales, dependiendo de su alcance. El depurador muestra cada array con un nivel jerárquico adicional; esto le permite obtener o cambiar los valores de los elementos del array, si los hay. El depurador muestra los primeros 100 elementos, incluyendo el elemento cero. El valor de la columna muestra el tamaño del array respecto a su nombre. Después de desplegar el array, el primer subelemento muestra el número del elemento seleccionado actualmente, luego el elemento cero, luego los otros elementos (hasta 100). Usted puede modificar los arrays Alfa, Texto, Numérico y Fecha. También es posible modificar el número de elemento seleccionado, el elemento cero y los otros elementos (hasta 100). No se puede modificar el tamaño del array.

Nota: en cualquier momento, usted puede arrastrar y soltar un elemento del panel de expresión al **Panel de evaluación**, incluyendo un elemento del array.

Valores del formulario actual

Este tema contiene el nombre de cada objeto dinámico incluido en el formulario actual, así como el valor de su variable asociada:



Algunos objetos, tales como arrays listbox, pueden presentarse como dos objetos distintos (la variable del objeto y su fuente de datos).

Esta lista es particularmente útil cuando se sus formularios utilizan variables dinámicas de manera intensiva:es fácil de identificar variables dinámicas a través de los nombres de objetos del formulario. Puede mostrar el nombre interno de las variables dinámicas seleccionando **Show Types** en el menú contextual:
Los nombres de las variables dinámicas son de la forma "$form.4B9.42":

Constantes

Muestra las constantes predefinidas de 4D, como en la página Constantes de la ventana del Explorador. Las expresiones de este tema no pueden modificarse.

Tablas y campos

Este tema lista las tablas y campos en la base de datos; no lista los subcampos. Para cada tabla, la columna valor muestra el tamaño de la selección actual para el proceso actual así como también (si la línea de la tabla está desplegada) el número de registros bloqueados. Para cada campo, la columna valor muestra el valor del campo (a excepción de imágenes, subtablas y BLOBs) para el registro actual, si lo hay. En este tema, los valores de los campos pueden modificarse (no es posible deshacer una acción), pero la información de la tabla no.

Semáforos

Lista los semáforos locales asignados actualmente. Para cada semáforo, la columna valor suministra el nombre del proceso que ha puesto el semáforo. Esta lista estará vacía si usted no utiliza semáforos. Las expresiones de este tema no pueden modificarse. No es posible visualizar los semáforos globales.

Conjuntos

Lista los conjuntos definidos en el proceso actual (el que usted está depurando actualmente); así como también los conjuntos interproceso. Para cada conjunto, la columna valor muestra el número de registros y el nombre de la tabla. Esta lista puede estar vacía si usted no utiliza conjuntos. Las expresiones de este tema no pueden modificarse.

Procesos

Lista los procesos iniciados desde el comienzo de la sesión de trabajo. La columna valor muestra el tiempo utilizado y el estado actual de cada proceso (por ejemplo, Ejecutándose, Suspendido, etc.). Las expresiones de este tema no pueden modificarse.

Selecciones temporales

Lista las selecciones temporales proceso definidas en el proceso actual (el que está depurando actualmente); también lista las selecciones interproceso. Para cada selección, la columna valor muestra el número de registros y el nombre de la tabla. Esta lista puede estar vacía si usted no utiliza las selecciones temporales. Las expresiones de este tema no pueden modificarse.

Información

Este tema muestra información general sobre el funcionamiento de bases, tal como la tabla por defecto actual (si existe), la memoria física, virtual, libre y usada, el destino de búsqueda, etc. Esta información le permite examinar el funcionamiento de la

base.

Estadísticas de la Caché

Este tema muestra información relativa al servidor web de la aplicación (sólo disponible si el servidor web está activo):

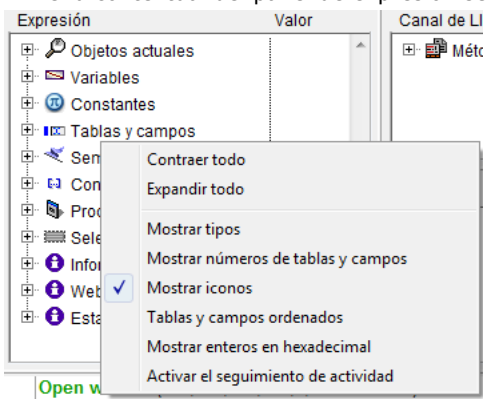
- Archivo web a enviar: nombre del archivo web en espera de ser enviado (si lo hay)
- Uso de la caché web: número de páginas presentes en la caché web y porcentaje de uso
- Tiempo de actividad del servidor web: tiempo de uso del servidor web en formato horas:minutos:segundos
- Número de peticiones http: número total de peticiones HTTP recibidas desde el lanzamiento del servidor web, así como también el número instantáneo de peticiones por segundo
- Número de procesos web activos: número de procesos web activos, todos los tipos de procesos web juntos.

Menú contextual del panel de expresión

El menú contextual del panel de expresión ofrece opciones adicionales. Para mostrar este menú:

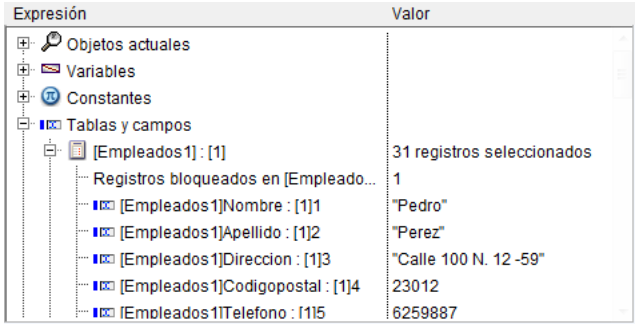
- En Windows, haga clic en cualquier parte del panel de expresión utilizando el botón **derecho** del ratón.
- En Macintosh, utilice la combinación Control-Clic en cualquier parte en el panel de expresión.

El menú contextual del panel de expresión se presenta a continuación:



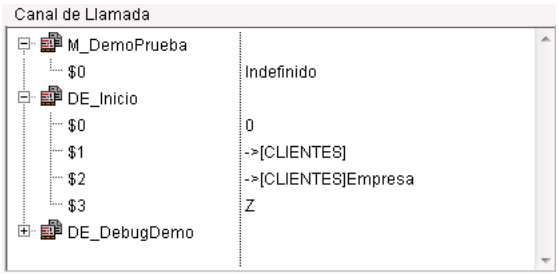
- **Contraer todo**: contrae todos los niveles de la lista jerárquica de las expresiones.
- **Expandir todo**: expande todos los niveles de la lista jerárquica de las expresiones.
- **Mostrar tipos**: muestra el tipo del objeto para cada objeto (cuando sea conveniente).
- **Mostrar números de tablas y campos**: muestra el número de cada tabla o campo de los **Campos**. Si usted trabaja con los números de las tablas o de los campos, o con punteros utilizando comandos como **Table** o **Field**, esta opción es muy útil.
- **Mostrar iconos**: muestra un icono que indica para cada objeto su tipo. Puede desactivar esta opción para acelerar la visualización, o simplemente porque prefiere utilizar la opción **Mostrar tipos**.
- **Tablas y campos ordenados**: hace que las tablas y campos aparezcan en orden alfabético, en sus listas respectivas.
- **Mostrar enteros en hexadecimal**: los números generalmente aparecen en notación decimal. Esta opción los muestra en notación hexadecimal. **Nota:** Para introducir un valor numérico en hexadecimal, digite 0x (cero + "x"), seguido por los dígitos hexadecimales.
- **Activar el seguimiento de actividad**.

La siguiente imagen muestra el panel de expresión con todas las opciones seleccionadas:



Panel del canal de llamada

Un método podría llamar otros métodos, que a su vez pueden llamar otros métodos. Por esta razón, es muy útil ver la cadena de métodos, o **Canal de llamada,** durante el proceso de depuración. El panel del canal de llamada, está en la parte superior derecha de la ventana del depurador. Los métodos en este panel aparecen de manera jerárquica. Este es un ejemplo del panel del canal de llamada:

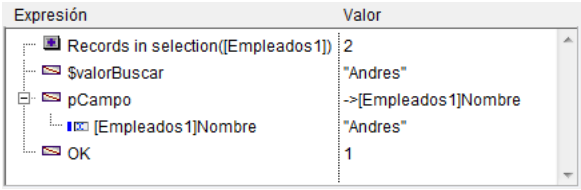


- Cada nivel principal es el nombre de un método. El elemento ubicado en la parte superior es el método que usted está depurando actualmente, el nivel siguiente es el nombre del método que llamó al método que está depurando actualmente, el nivel siguiente es la llamada del método llamante, etc. En el ejemplo anterior, se ha rastreado el método **M_BitTestDemo**; ha sido llamado por el método **DE_Initialize**, el cual ha sido llamado por **DE_DebugDemo**.
- Al hacer doble clic en el nombre de un método en el usted va al método que lo llamó, cuyo código fuente aparece en el **Panel de evaluación de métodos**. Al hacer esto, usted puede ver rápidamente "cómo" el método llamante efectúo su llamada al método llamado. De esta forma, puede examinar todas las etapas de la cadena de llamada.
- Al hacer clic en el icono junto a un nombre de método se despliega o contrae la lista de parámetros ($1, $2...) y el resultado (*$0*) opcional de una función. Los valores aparecen al lado derecho del panel. Haciendo clic en cualquier valor del lado derecho puede cambiar el valor del resultado de la función o del parámetro. En la imagen anterior:
  1. **M_BitTestDemo** no ha recibido ningún parámetro.
  2. **M_BitTestDemo** actualmente está indefinido, porque el método no asignó ningún valor a *$0* (porque no ha ejecutado esta asignación aún o porque el método es una subrutina y no una función).
  3. **DE_Initialize** ha recibido tres parámetros de **DE_DebugDemo**. *$1* ies un puntero a la tabla *[Clientes*], *$2* es un puntero al campo *[Clientes]Empresa* y *$3* es un parámetro alfanumérico cuyo valor es *"Z"*.
- Después de desplegar la lista de parámetros de un método, puede también arrastrar y soltar parámetros y resultados de función al **Panel de evaluación**.

Panel de evaluación

Justo debajo del **Panel del canal de llamada** se encuentra el **Panel de evaluación**. Este panel se utiliza para evaluar expresiones. Todo tipo de expresión puede evaluarse, incluyendo campos, variables, punteros, cálculos, funciones integradas, sus propias funciones y todo lo que retorne un valor.

Puede evaluar cualquier expresión que pueda mostrarse en forma de texto. Esto no aplica ni a los campos ni a las variables de tipo imagen o BLOB. Por otra parte, el depurador utiliza listas jerárquicas desplegadas para permitirle mostrar arrays y punteros. Para mostrar los contenidos de BLOBs, puede utilizar los comandos de BLOBs, como **BLOB to text**.
En el siguiente ejemplo, puede ver varios de estos elementos: dos variables, un puntero a un campo, el resultado de una función integrada y un cálculo.



Insertar una nueva expresión

Puede añadir una expresión a evaluar en el panel de expresión de una de las siguientes maneras:

- Arrastrando y soltando un objeto o expresión del **Panel de expresión**
- Arrastrando y soltando un objeto o expresión del **Panel del canal de llamada**
- En el **Panel de evaluación de métodos**, haga clic en una expresión que pueda ser evaluada

Para crear una expresión vacía, haga doble clic en alguna parte en el espacio vacío del panel de evaluación. Esto añade una expresión ` *Nueva expresión* y luego pasa al modo de edición para usted pueda modificarla. Puede introducir cualquier fórmula 4D que devuelva un resultado.

Después de introducir la fórmula, presione la tecla **Intro** o **Retorno de carro** (o haga clic en alguna parte del panel) para evaluar la expresión.

Para cambiar la expresión, haga clic en ella para seleccionarla, luego haga clic nuevamente (o presione Enter, teclado numérico) para ir al modo edición.

Si no necesita más una expresión, haga clic en ella para seleccionarla, luego presione **Retroceso** o **Suprimir**.

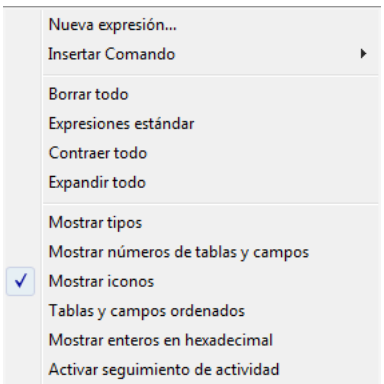
Advertencia: sea cuidadoso cuando evalúa una expresión 4D modificando el valor de una de las *variables sistema* (por ejemplo, la variable OK) porque la ejecución del resto del método puede alterarse.

Menú contextual del panel de evaluación

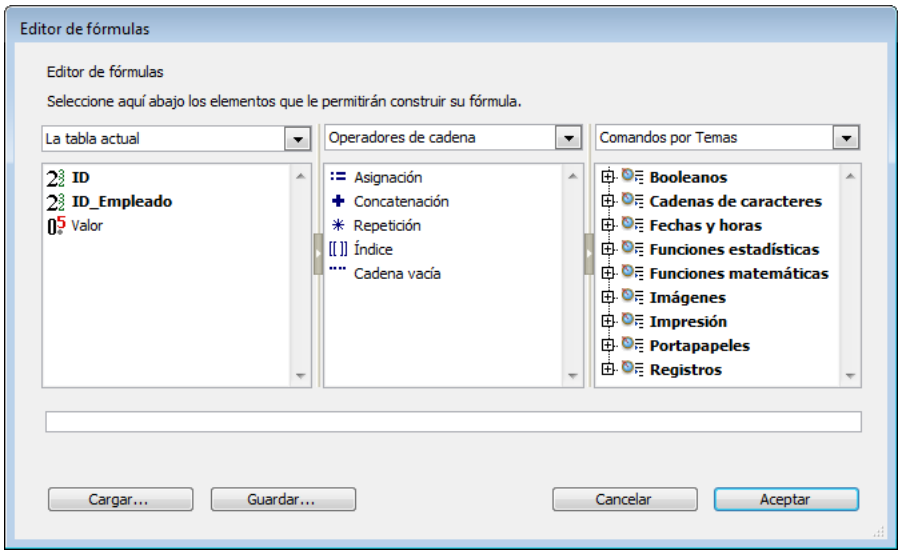
Para ayudarle a introducir y editar una expresión, el menú contextual del panel de evaluación le da acceso al editor de fórmulas de 4D. De hecho, el menú contextual también le ofrece opciones adicionales.

Para obtener este menú:

- En Windows, haga clic en cualquier parte del panel de evaluación utilizando el botón **derecho** del ratón
- En Macintosh, Control-Clic en cualquier parte del panel de evaluación.



- **Nueva expresión...**: este comando inserta una nueva expresión y muestra el editor de fórmulas de 4D (como se muestra en la imagen) de manera que usted pueda editar la nueva expresión.

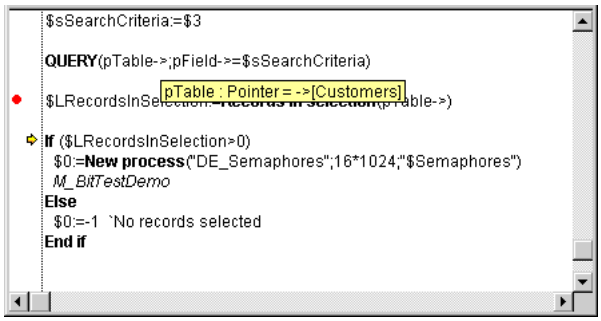


Para mayor información sobre el editor de fórmulas, consulte el Manual de diseño.

- **Insertar comando**: este elemento del menú jerárquico es un atajo para insertar un comando como una nueva expresión, sin utilizar el editor de fórmulas.
- **Borrar todo**: borra todas las expresiones presentes.
- **Expresiones estándar**: recopia la lista de objetos en el área de expresión.
- **Contraer/Expandir todo**: contrae o expande todas las expresiones cuya evaluación se realiza utilizando listas jerárquicas (punteros, arrays, etc.)
- **Mostrar tipos**: muestra el tipo para cada objeto (cuando sea pertinente).
- **Mostrar números de tablas y campos**: muestra el número de cada tabla o campo de **Campos**. Si trabaja con los números de tablas o de los campos o con punteros utilizando comandos como **Table** o **Field**, esta opción es muy útil.
- **Mostrar iconos**: muestra un icono que simboliza el tipo para cada objeto. Puede desactivar esta opción para acelerar la visualización, o simplemente utilizar la opción **Mostrar tipos**.
- **Tablas y campos ordenados**: la tabla y los campos aparecen en orden alfabético, dentro de sus listas respectivas.
- **Mostrar enteros en hexadecimal**: los números aparecen en notación decimal. Esta opción los muestra en notación hexadecimal. **Nota:** para introducir un valor numérico en hexadecimal, digite 0x (cero + "x"), seguido por los dígitos hexadecimales.
- **Activar seguimiento de actividad**: activa y muestra información sobre el monitoreo de la actividad (ver la sección **Panel de evaluación**).

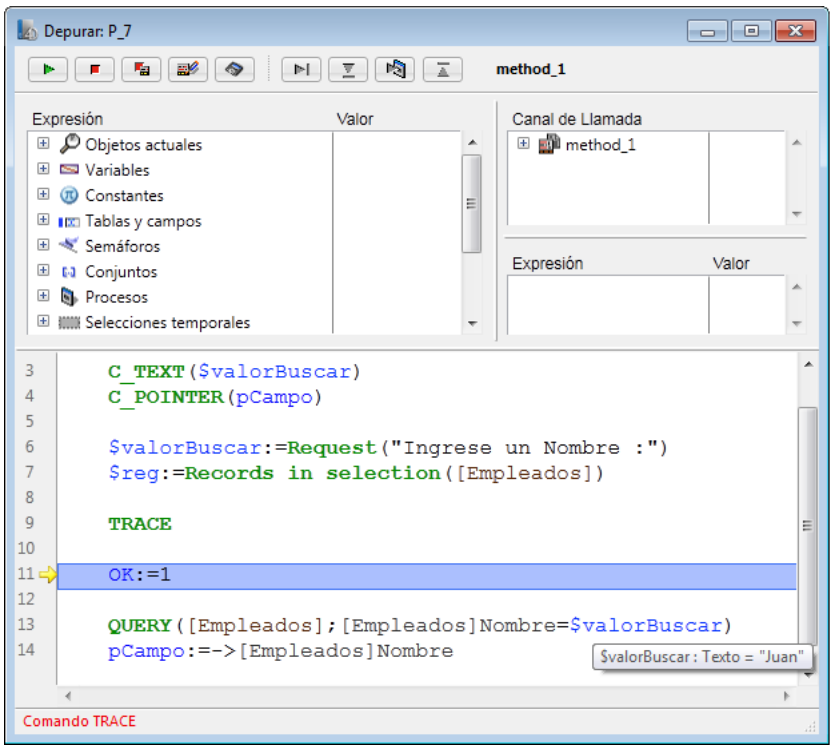
Panel de evaluación de métodos

El muestra el código fuente del método que está siendo depurado.
Si el método es demasiado largo para el área de texto, puede desplazarse para ver otras partes del método.
Al mover el puntero del ratón sobre cualquier expresión que pueda ser evaluada (campo, variable, puntero, array,...) hará que aparezca un **Mensaje** mostrando el valor actual del objeto o expresión y su tipo declarado. Por ejemplo:



Aparece un mensaje porque el puntero del ratón se ubica sobre la variable *pTable*, según el mensaje, es un puntero a la tabla *[Customers]*.

- Igualmente puede seleccionar una porción de texto en el área de visualización del código en ejecución. En este caso, cuando el cursor se ubica sobre el texto seleccionado, un mensaje muestra el valor de la selección:



Cuando hace clic en un nombre de variable o campo, se selecciona automáticamente.

Consejo: es posible copiar toda expresión seleccionada (que pueda evaluarse) en el **Panel de evaluación de métodos** al **Panel de evaluación**. Puede hacerlo de estas maneras:

- simplemente arrastrando y soltando: haga clic en el texto seleccionado, arrástrelo y suéltelo en el área de evaluación.
- utilizando la combinación de teclas **Ctrl+D** (Windows) o **comando+D** (Mac OS).

Contador del programa

Una flecha amarilla al margen izquierdo del panel de evaluación de métodos (ver la imagen anterior) indica cual es la próxima línea a ejecutar. Esta flecha se llama **contador del programa**. El contador del programa siempre indica la línea que está a punto de ser ejecutada.

Nota: por defecto, la línea contador de programa (también llamada la *línea de ejecución*) se resalta en el depurador. Puede personalizar el color del resaltado en **Página Métodos** de las Preferencias.

Por propósitos de depuración, puede **cambiar** el contador del programa para el método que se encuentra en la parte superior del canal de llamada (el método que se está ejecutando). Para hacer esto, simplemente haga clic y arrastre la flecha amarilla verticalmente, hacia la línea que quiera.

Advertencia: ¡Utilice esta característica con cuidado!

Mover el contador del programa hacia adelante NO significa que el depurador ejecute rápidamente las líneas por las que está pasando. De la misma forma, mover el contador del programa hacia atrás NO significa que el depurador reverse el efecto de las líneas que ya han sido ejecutadas.

Al mover el contador del programa, usted indica al depurador simplemente que "continúe la depuración o ejecución desde este punto." Todos los parámetros, campos, variables, etc. actuales no se ven afectados por el desplazamiento.

Este es un ejemplo de desplazamiento del contador del programa. Supongamos que está depurando el siguiente código:

```
` ...
If(Esta condición)
   HACER ALGO
Else
   HACER ALGO MÁS
End if
` ...
```

El contador del programa está ubicado en la línea **If (Esta condición)**. Usted avanza un paso y ve que el contador del programa se mueve hacia la línea **HACER ALGO MÁS**. Usted quería ejecutar la otra alternativa. En este caso, y en la medida en que la expresión **Esta condición** no efectúa operaciones que afecten las etapas siguientes de su prueba, mueva el contador del programa a la línea **HACER ALGO**. Ahora puede continuar depurando la parte del código que le interesa.

Definir los puntos de interrupción en el depurador

En el proceso de depuración, usted podría necesitar saltarse algunas partes del código. El depurador le ofrece varios métodos para ejecutar código **hasta cierto punto**:

- Mientras ejecuta paso a paso, puede hacer clic en el botón **Ejecutar paso a paso** en lugar del botón **Ejecutar y entrar**. Esto es útil cuando usted no quiere entrar en posibles subrutinas o funciones llamadas en la línea del contador del programa.
- Si por error entró a una subrutina, puede ejecutarla e ir directamente al método que la llamó haciendo clic en el botón **Ejecutar y salir**.
- Si en algún punto tiene una llamada al comando **TRACE**, puede hacer clic en el botón **Reanudar,** el cual reasume la ejecución hasta la llamada al comando.

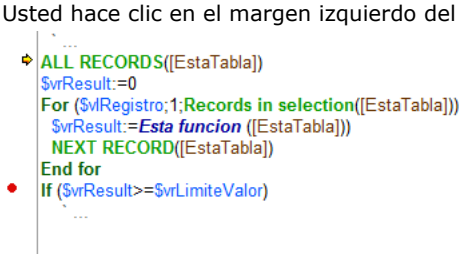
Por ejemplo, imagine que está ejecutando el siguiente código. El contador del programa está ubicado en la línea *ALL RECORDS([EstaTabla])*:

```
` ...
ALL RECORDS([EstaTabla])
$vrResult:=0
For($vlRegistro;1;Records in selection([EstaTabla]))
   $vrResult:=Esta Funcion([EstaTabla])
   NEXT RECORD([EstaTabla])
End for
If($vrResult>=$vrLimitValor)
` ...
```

Su meta es evaluar el valor de *$vrResult* después de que se haya completado el bucle For. Como toma bastante tiempo alcanzar este punto en su código, usted no quiere abortar la ejecución actual, luego edite el método con el fin de insertar un llamada a **TRACE** antes de la línea *If ($vrResult....*

Una solución es ir paso a paso a través del bucle, sin embargo, si la tabla *[EstaTabla]* contiene muchos registros, usted va a gastar todo un día en esta operación. En este tipo de situación, el *depurador* le ofrece **puntos de interrupción**. Puede insertar puntos de interrupción haciendo clic en el margen izquierdo del panel de evaluación de métodos.

Por ejemplo:
Usted hace clic en el margen izquierdo del panel de evaluación de métodos a nivel de la línea *If ($vrResult...*:



Esto inserta un punto de interrupción para la línea. El punto de interrupción se indica por un punto rojo. Luego haga clic en el botón **Reanudar**.

Esto retoma la ejecución normal **hasta la línea** marcada con el punto de interrupción. Esa línea no se ejecuta, usted vuelve al modo depuración. En este ejemplo, todo el bucle ha sido ejecutado normalmente. Luego cuando se alcanza el punto de interrupción, usted sólo necesita mover el botón del ratón sobre *$vrResult* para evaluar su valor en el punto de salida del bucle.

Colocar un punto de interrupción más allá del contador del programa y hacer clic en el botón **Reanudar** le permite evitar depurar **partes** del método.

Nota: también puede definir puntos de interrupción directamente en el editor de métodos de 4D. Consulte la sección **Puntos de interrupción**.

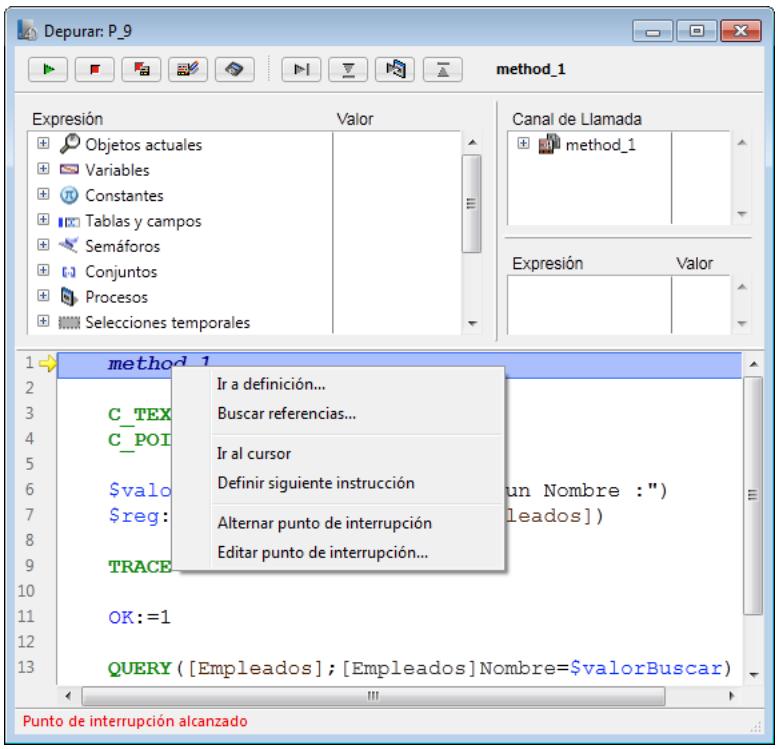
Una vez añada un punto de interrupción, permanece asociado al método. Incluso si sale de la base y entra nuevamente más tarde, el punto de interrupción estará ahí.

Hay dos formas de eliminar un punto de interrupción persistente:

- Si usted terminó de utilizarlo, simplemente haga clic en el punto rojo, el punto de interrupción desaparece.
- Si no ha terminado de utilizarlo totalmente, podría querer conservarlo. Puede desactivarlo temporalmente editándolo. Esto se explica en la sección **Puntos de interrupción**.

Menú contextual de la ventana de evaluación de los métodos

El menú contextual de **Panel de evaluación de métodos** da acceso a varias funciones que son útiles durante la ejecución de métodos en modo de Rastreo:



- **Ir a definición**: permite acceder a la definición del objeto seleccionado. Este comando está disponible para los siguientes objetos:
  - *Métodos proyecto*: muestra el contenido del método en una nueva ventana del editor de métodos.
  - *Campos*: muestra las propiedades del campo en el inspector de la ventana de estructura,
  - *Tablas*: muestra las propiedades de la tabla en el inspector de la ventana de la estructura,
  - *Formularios*: la forma Muestra en el editor de formularios,
  - *Variables* (local, proceso, interproceso o parámetro $): muestra la línea de declaración de la variable en el método actual o en los métodos compilador.
- **Buscar referencias...**(también disponible en el editor de métodos): busca todos los objetos de la base (métodos y formularios) en la que el elemento actual es referenciado. El elemento actual es el elemento seleccionado o el elemento en el cual se encuentra el cursor. Este puede ser el nombre de un campo, variable, comando, cadena, etc. El resultado de la búsqueda se muestra en una nueva ventana de resultados estándar.
- **Ir al cursor**: ejecuta la instrucciones entre el contador de programa (flecha amarilla) y la línea seleccionada del método (donde se encuentra el cursor).
- **Definir siguiente instrucción**: mueve el contador del programa hasta la línea seleccionada sin ejecutar esta línea o las líneas intermedias. La línea designada sólo se ejecuta si el usuario hace clic en uno de los botones de ejecución.
- **Alternar punto de interrupción** (también disponible en el editor de métodos): permite alternativamente insertar o eliminar el punto de interrupción correspondiente a la línea seleccionada. Esta función modifica el punto de interrupción permanente: por ejemplo, si elimina un punto de interrupción en el depurador, ya no aparece en el método original.
- **Editar punto de interrupción** (también disponible en el editor de métodos): muestra la caja de diálogo de definición de Propiedades del punto de interrupción. Esta función modifica el punto de interrupción permanente.

Puntos de interrupción

Como se explicó en la sección **Std deviation**, usted define un punto de interrupción haciendo clic en el margen izquierdo de la ventana de evaluación de métodos o en la ventana del editor de métodos, a nivel de la línea de código en la cual usted quiere crear la interrupción.

Nota: como puede insertar, modificar o borrar puntos de interrupción en el editor de métodos o el depurador, hay una interacción dinámica entre los dos editores (así como también con el explorador de ejecución) con relación a los puntos de interrupción.

En la siguiente imagen, un punto de interrupción ha sido definido, en el depurador, en la línea **If($vrResult>=$vrLimitValor)**:

```
  ...
ALL RECORDS([EstaTabla])
$vrResult:=0
For ($vlRegistro;1;Records in selection([EstaTabla]))
  $vrResult:=Esta funcion ([EstaTabla])
  NEXT RECORD([EstaTabla])
End for
If ($vrResult>=$vrLimiteValor)
  ...
```

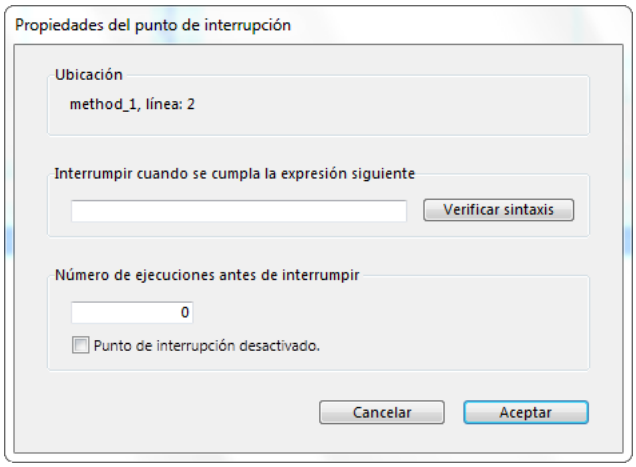
Si hace clic nuevamente en el punto rojo, elimina el punto de ruptura.

Edición de un punto de interrupción

Puede acceder a la ventana **Propiedades del punto de interrupción** seleccionando el comando **Modificar punto de interrupción** en el menú contextual de la **Panel de evaluación de métodos** o presionando Alt-clic (Windows) u Opción-clic (Macintosh) en la margen izquierda de la **Std deviation** (o del editor de métodos).

- Si hace clic en un punto de interrupción existente, la ventana se abre para ese punto de interrupción.
- Si hace clic en una línea donde no hay ningún punto de interrupción, el depurador crea uno y muestra la ventana para el nuevo punto de interrupción.

Esta es la ventana de **Propiedades del punto de interrupción**:



Estas son las propiedades:

Ubicación: indica el nombre del método y el número de la línea dónde está ubicado el punto de interrupción. No es posible modificar esta información.

Tipo: por defecto, el depurador le permite crear puntos de interrupción **persistentes**, representados por un punto rojo en el de la ventana del depurador. Para crear un punto de interrupción temporal, seleccione la opción **Temporal**. Un punto de interrupción temporal es útil cuando usted quiere interrumpir sólo una vez en un método. Un punto de interrupción temporal se identifica con un punto verde en el **Panel de evaluación de métodos** de la ventana del depurador. Igualmente puede definir un punto de interrupción temporal directamente en el **Panel de evaluación de métodos** haciendo clic en el margen izquierdo mientras presiona Alt+Mayús (Windows) u Opción+Mayús (Macintosh).

Nota: los puntos de interrupción temporales pueden definirse en el depurador únicamente.

Interrumpir cuando se cumpla la siguiente impresión: puede crear puntos de interrupción condicionales introduciendo una fórmula 4D que devuelva Verdadero o Falso. Por ejemplo, si quiere interrumpir en una línea sólo cuando *Records in selection([aTabla])=0*, introduzca esta fórmula, y la interrupción ocurrirá sólo si no hay registros seleccionados para la tabla *[aTabla]*, cuando el depurador encuentre la línea con este punto de interrupción. Si no está seguro de la sintaxis de su formula, haga clic en el botón **Verificar sintaxis**.

Número de ejecuciones antes de interrumpir: puede definir un punto de interrupción para una línea de código ubicada en una estructura de bucle (While, Repeat, o For) o ubicada en una subrutina o función llamada desde un bucle. Por ejemplo, si sabe que el "problema" que está buscando no ocurre antes de las primeras 200 iteraciones del bucle. Introduzca 200 y el punto de interrupción se activará en la iteración 201.

Punto de interrupción desactivado: si actualmente no necesita un punto de interrupción, pero podría necesitarlo más adelante, puede desactivarlo temporalmente editándolo. Un punto de interrupción desactivado aparece como un guión (-) en lugar de un punto (•) en el editor de métodos o en la lista de puntos de interrupción.

Usted crea y edita puntos de interrupción desde la ventana del depurador y del editor de métodos. Igualmente puede editar los puntos de interrupción utilizando la página Punto de interrupción del Explorador de ejecución. Para mayor información, consulte la sección **Month of**.

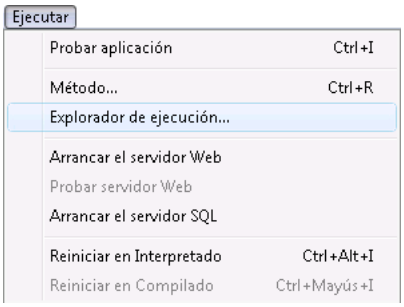
Lista de puntos de interrupción

La Lista de puntos de interrupción es una página del Explorador de ejecución que le permite administrar los puntos de interrupción creados en la ventana del depurador o en el editor de métodos.

Para abrir la página Lista de puntos de interrupción:

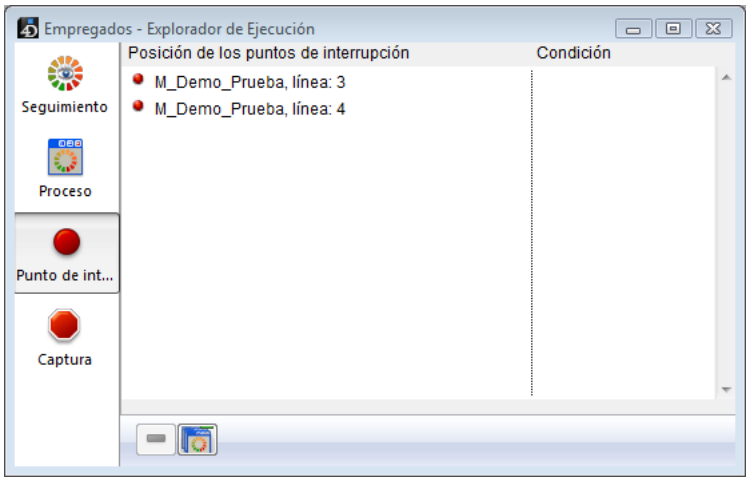
1. Elija **Explorador de ejecución** en el menú **Ejecutar**.

El explorador de ejecución puede aparecer bajo la forma de una paleta flotante. En este caso, la paleta flotante permanece en el primer plano. Para hacer esto, mantenga presionada la tecla **Mayús** mientras selecciona el **Explorador de ejecución** en el menú **Ejecutar**. El Explorador Runtime está disponible en todos los entornos 4D. Para mayor información por favor consulte el manual de Diseño.



Aparece la ventana del explorador de ejecución.

2. Haga clic en la pestaña **Punto de interrupción** para mostrar la lista de puntos de interrupción:



La lista de puntos de interrupción está compuesta por dos columnas:

- La columna de la izquierda indica el estado activo/inactivo de los puntos de interrupción, seguido por el nombre del método y el número de la línea donde está ubicado el punto de interrupción (utilizando la ventana del Depurador o el editor de métodos).
- La columna a la derecha muestra la condición asociada con el punto de interrupción, si la hay.

Utilizando esta ventana, usted puede:

- Definir una condición para un punto de interrupción,
- Habilitar, deshabilitar o eliminar cada punto de interrupción,
- Abrir una ventana del editor de métodos mostrando el método al que está asociado el punto de interrupción, haciendo doble clic en el punto de interrupción.

Sin embargo, no es posible añadir un nuevo punto de interrupción persistente desde esta ventana. Los puntos de interrupción persistentes sólo pueden ser creados desde la ventana del depurador o desde el editor de métodos.

Definir una condición para un punto de interrupción

Para definir una condición para un punto de interrupción, proceda de la siguiente manera:

1. Haga clic en la entrada en la columna de la derecha
2. Introduzca una fórmula 4D (expresión o llamada de comando o método de proyecto) que devuelva un valor booleano.

Nota: Para suprimir una condición, borre la fórmula correspondiente.

Habilitar/Deshabilitar un punto de interrupción

Para habilitar o deshabilitar un punto de interrupción:

1. Seleccione el punto de interrupción haciendo clic en él o utilizando las flechas para navegar por la lista (si la entrada seleccionada no está en modo edición).

2. Haga clic en el botón **Habilitar/Deshabilitar** en el menú contextual.

Atajo: Cada punto de interrupción en la lista puede ser habilitado/deshabilitado haciendo clic directamente en el punto (•). El punto se convierte en guión (–) cuando está deshabilitado.

Eliminar un punto de interrupción

Para eliminar un punto de interrupción:

1. Seleccione el punto de interrupción haciendo clic en él o utilizando las flechas para navegar por la lista.
2. Presione la tecla **Supr** o haga clic en el botón **Retroceso**.

Nota: para eliminar todos los puntos de interrupción, haga clic en el botón **Eliminar todos** o seleccione **Borrar todos** en el menú contextual.

Consejos:

- La adición de condiciones a los puntos de interrupción vuelve lenta la ejecución del código, porque la condición tiene que evaluarse cada vez que se encuentra el punto de interrupción. Por otra parte, añadir condiciones acelera el proceso de depuración, porque automáticamente se ignoran las ocurrencias que no cumplen las condiciones.
- La deshabilitación de un punto de interrupción tiene casi el mismo efecto que borrarlo. Durante la ejecución, el depurador casi no pasa tiempo en el punto de interrupción. La ventaja de deshabilitar un punto de interrupción es que usted no tiene que volver a definir el punto de interrupción en caso de necesitarlo nuevamente.

Capturas de comandos

La **lista de comandos capturados** es una página del explorador de ejecución que le permite añadir puntos de interrupción adicionales a su código interceptando las llamadas a los comandos 4D.

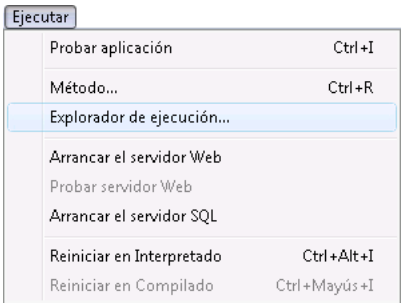
Colocar un punto de interrupción sobre un comando le permite comenzar a depurar la ejecución de cualquier proceso tan pronto como un comando en particular sea llamado por el proceso. A diferencia de un punto de interrupción colocado en un método de proyecto en particular (y que por lo tanto activa una depuración sólo cuando es alcanzado), el alcance de un punto de interrupción sobre un comando incluye todos los procesos que ejecutan el código 4D y llaman a ese comando.

Colocar un punto de interrupción sobre un comando es una forma conveniente de depurar grandes porciones de código sin definir puntos de interrupción en ubicaciones arbitrarias. Por ejemplo, si un registro que no debe ser borrado es borrado después de ejecutar uno o varios procesos, usted puede tratar de reducir el campo de investigación colocando un punto de interrupción sobre los comandos tales como **DELETE RECORD** y **DELETE SELECTION**. Cada vez que estos comandos son llamados, puede revisar si el registro en cuestión ha sido borrado y aislar la parte defectuosa del código.

Con alguna experiencia, puede combinar el uso de puntos de interrupción en los métodos y sobre los comandos.

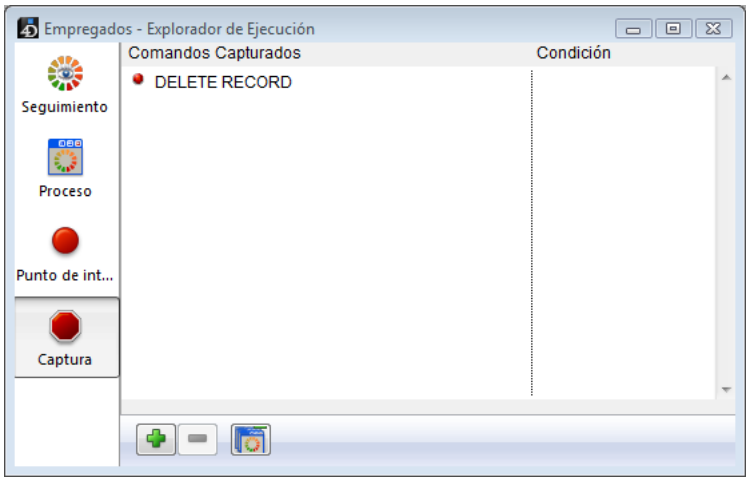
Para abrir la lista de comandos capturados:

1. Elija **Explorador de ejecución** en el menú **Ejecutar**.
El explorador de ejecución puede aparecer bajo la forma de una paleta flotante. En este caso, la paleta flotante permanece en el primer plano. Para hacer esto, mantenga presionada la tecla **Mayús** mientras selecciona el **Explorador de ejecución** en el menú **Ejecutar**. El Explorador Runtime está disponible en todos los entornos 4D. Para mayor información por favor consulte el manual de Diseño.



Aparece la ventana del Explorador de ejecución.

2. Haga clic en la pestaña **Captura** para mostra la lista de comandos capturados:



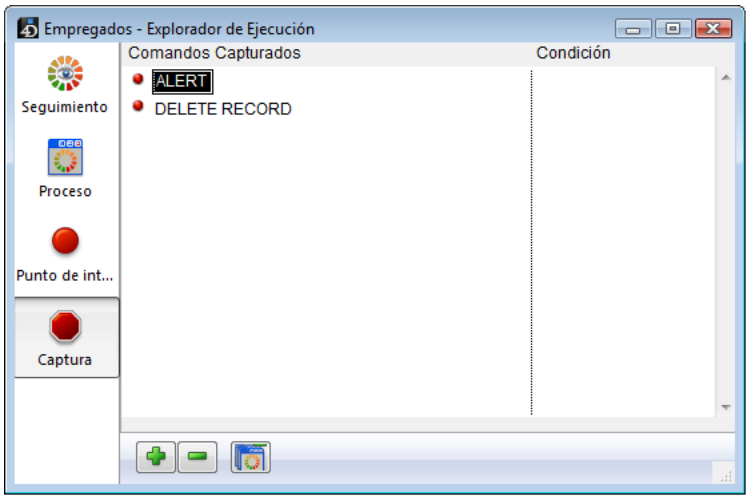
Esta página lista los comandos a capturar durante la ejecución. Esta compuesta por dos columnas:

- La columna de la izquierda muestra el estado Habilitado/Deshabilitado del comando a capturar, seguido por el nombre del comando.
- La columna de la derecha muestra la condición asociada con el comando a capturar, si la hay.

Adición de un nuevo comando a capturar

Para añadir un nuevo comando:

1. Haga clic en el botón **Añadir** (en forma de +) ubicado debajo de la lista. se añade una nueva entrada a la lista con el comando **ALERT** por defecto.



Luego puede hacer clic en la etiqueta **ALERT** e introduzca el nombre del comando a capturar. Una vez haya terminado, presione **Intro** o **Retorno de carro** para validar su elección.

Edición del nombre del comando a capturar

Para editar el comando a capturar:

1. Seleccione el comando haciendo clic en él o utilizando las teclas de flecha para navegar por la lista (si la entrada seleccionada actualmente no está en modo edición).
2. Para pasar una entrada de modo edición a modo selección y viceversa, presione **Intro** o **Retorno de carro**.
3. Introduzca o modifique el nombre del comando.
4. Para validar sus cambios, presione **Intro** o **Retorno de carro**.

Activar/Desactivar un comando a capturar

Para habilitar o deshabilitar un comando a capturar:

1. Haga clic haciendo clic en el punto (•) ubicado en frente de la etiqueta del comando.
   Esto le permite activar alternativamente y/o desactivar el punto de ruptura. El color del punto indica su estado:
   - rojo = activado,
   - naranja = desactivado.

Nota: el desactivar un punto de ruptura tiene casi el mismo efecto que eliminarlo. Durante la ejecución, el depurador pasa muy poco tiempo en la entrada. La ventaja de la desactivación de una entrada es que usted no tiene que volver a crearla cuando la necesite de nuevo.

Eliminar un comando a capturar

Para eliminar un comando a capturar:

1. Seleccione la entrada haciendo clic en ella o utilizando las teclas de flecha para navegar por la lista (si la entrada seleccionada actualmente no está en modo edición).
2. Presione **Retroceso** o **Supr** o haga clic en el botón **Eliminar** ubicado debajo de la lista.
3. Para eliminar todos los comandos a capturar, haga clic en el botón **Eliminar todo**.

Definir una condición para un comando a capturar

Para definir una condición para un comando a capturar:

1. Haga clic en la entrada de la columna derecha.
   Aparece un cursor de entrada.
2. Introduzca una fórmula 4D (expresión, llamada a un comando o método de proyecto) que devuelva un valor booleano.

Nota: para eliminar una condición, borre su fórmula.

La adición de condiciones permite detener la ejecución cuando se invoca el comando sólo si se cumple la condición. Por ejemplo, si asocia la condición "**Records in selection**([Emp]>10)" con el punto de ruptura en el comando **DELETE SELECTION**, el código no se detendrá durante la ejecución del comando **DELETE SELECTION** si la selección actual de la tabla [Emp] sólo contiene 9 registros (o menos).

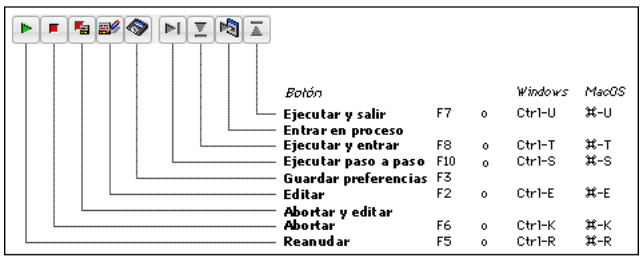
La adición de condiciones a los comandos capturados ralentiza la ejecución, ya que la condición tiene que ser evaluada cada vez que se encuentra con una excepción. Por otra parte, la adición de condiciones acelera el proceso de depuración, porque 4D salta automáticamente ocurrencias que no coinciden con las condiciones.

Atajos del depurador

Esta sección lista todos los atajos disponibles en la ventana del depurador.

Barra de herramientas del control de ejecución

La siguiente imagen muestra los atajos de los nueve botones ubicados en la parte superior izquierda de la ventana del depurador:



La combinación Mayús +F5 o Mayús+clic en el botón **Reanudar** retoma la ejecución y desactiva todas las llamadas posteriores a **TRACE** en el proceso actual.

Panel de expresión

- Un clic con el botón derecho del ratón (Windows) o Control-Clic (Macintosh) en el **Panel de expresión** despliega el menú contextual.
- Un doble clic en un elemento del **Panel de expresión** copia el elemento en el **Panel de evaluación**.

Panel del canal de llamada

- Un doble clic en el nombre de un método en el **Panel del canal de llamada** muestra el método en el **Panel de evaluación** en la línea correspondiente a la cadena de llamada.

Panel de evaluación

- Un clic con el botón derecho del ratón (Windows) o Control-Clic (Macintosh) en el **Panel de evaluación** despliega el menú contextual del panel.
- Un doble clic en la ventana de evaluación crea una nueva expresión.

Panel de evaluación de los métodos

- Un clic en el margen izquierdo define (persistente) o elemina un punto de interrupción.
- **Alt-Mayús-Clic** (Windows) u **Opción-Mayús Clic** (Macintosh) define un punto de interrupción temporal.
- **Alt-Clic** (Windows) u **Opción-Clic** muestra la ventana de Propiedades del punto de interrupción para un punto de interrupción nuevo o existente.
- Una expresión u objeto seleccionado puede copiarse en el **Panel de evaluación** simplemente arrastrando y soltando
- Las combinaciones **Ctrl+D** (Windows) o **Comando+D** (Mac OS) copian el texto seleccionado en el **Panel de evaluación**.

Todos los paneles

- **Ctrl + *** (Windows) o **Comando + *** (Mac OS) fuerzan la actualización del **Panel de expresión**.
- Cuando ningún objeto está seleccionado en los paneles, digitado **Intro** avanza una línea.
- Cuando el valor de un elemento es seleccionado, utilice las teclas de flecha para navegar por la lista.
- Cuando esté editando un elemento, utilice las teclas de flecha para mover el cursor; utilice **Ctrl-A/X/C/V** (Windows) o **Comando-A/X/C/V** (Macintosh) como atajos para los comandos de menú **Edición**: **Seleccionar todo/Cortar/Copiar/Pegar**.

Acceso a los objetos de desarrollo

Comandos del tema Acceso objetos diseño
Current method path
FORM GET NAMES
METHOD Get attribute
METHOD GET ATTRIBUTES
METHOD GET CODE
METHOD GET COMMENTS
METHOD GET FOLDERS
METHOD GET MODIFICATION DATE
METHOD GET NAMES
METHOD Get path
METHOD GET PATHS
METHOD GET PATHS FORM
METHOD OPEN PATH
METHOD RESOLVE PATH
METHOD SET ACCESS MODE
METHOD SET ATTRIBUTE
METHOD SET ATTRIBUTES
METHOD SET CODE
METHOD SET COMMENTS

Comandos del tema Acceso objetos diseño

4D permite acceder por programación al contenido de los métodos de sus aplicaciones. Este source toolkit facilita la integración de sus aplicaciones a las herramientas de control de código, en particular las aplicaciones de gestión de versiones (VCS). También le permite implementar sistemas avanzados de documentación del código, construir un explorador personalizado u organizar copias de seguridad programadas del código en forma de archivos de disco.

Los principios siguientes aplican:

- Cada método y formulario de una aplicación 4D tiene su propia dirección en forma de ruta de acceso. Por ejemplo, el método trigger de la tabla 1 es accesible en la dirección "[trigger]/table_1". Cada ruta de acceso de objeto es única en una aplicación.
  Nota: para garantizar la unicidad de las rutas de acceso, 4D no permite crear objetos con el mismo nombre en páginas formularios diferentes. En bases de datos convertidas de versiones anteriores a 4D v13, el CSM detecta los nombres duplicados.
- Puede acceder a los objetos de la aplicación 4D utilizando los comandos de este tema, por ejemplo **METHOD GET NAMES** o **METHOD GET PATHS**.
- La mayoría de los comandos en este tema funcionan en modo interpretado y en modo compilado. Sin embargo, los comandos que modifican las propiedades o acceden al contenido ejecutable de los métodos sólo pueden utilizarse en modo interpretado (ver la tabla abajo).
- Puede utilizar todos los comandos de este tema con 4D en modo local o remoto. Sin embargo, recuerde que no puede utilizar ciertos comandos en modo compilado: su propósito es crear herramientas personalizadas de ayuda de desarrollo. No debe utilizarlos para modificar dinámicamente el funcionamiento de una base en ejecución. Por ejemplo, no puede utilizar **METHOD SET ATTRIBUTE** para cambiar un atributo de método en función del estado del usuario actual.
- Cuando un comando de este tema se ejecuta desde un componente, accede por defecto a los objetos del componente. En este caso, para acceder a los objetos de la base local, pase * como último parámetro. Note que en este contexto, esta sintaxis sólo es posible para los comandos que modifican objetos (tales como **METHOD SET ATTRIBUTE**), ya que los componentes siempre se ejecutan en modo sólo lectura.

Uso en modo compilado

Por razones relacionadas con el mismo principio de procesos de compilación, sólo ciertos comandos de este tema son utilizables en modo compilado. La siguiente tabla indica la disponibilidad de los comandos en modo compilado:

Comando	Puede utilizarse en modo compilado
Current method path	Sí
FORM GET NAMES	Sí
METHOD SET ATTRIBUTE	No (*)
METHOD SET ATTRIBUTES	No (*)
METHOD SET CODE	No (*)
METHOD SET COMMENTS	No (*)
METHOD SET ACCESS MODE	Sí
METHOD Get attribute	Sí
METHOD GET ATTRIBUTES	Sí
METHOD Get path	Sí
METHOD GET PATHS	Sí
METHOD GET PATHS FORM	Sí
METHOD GET CODE	No (*)
METHOD GET COMMENTS	Sí
METHOD GET MODIFICATION DATE	Sí
METHOD GET FOLDERS	Sí
METHOD GET NAMES	Sí
METHOD OPEN PATH	No (*)
METHOD RESOLVE PATH	Yes

(*) El error -9762 "The command cannot be executed in a compiled database." se genera cuando el comando se ejecuta en modo compilado.

Creación de rutas de acceso

Por defecto, ningún archivo es creado en el disco por 4D. Sin embargo, las rutas de acceso generadas para los objetos son compatibles con la gestión de archivos del sistema operativo, ya que pueden ser utilizadas directamente para generar archivos en disco vía sus propios métodos de importación/exportación.

Más específicamente, los caracteres prohibidos tales como ":" están codificados en los nombres de los métodos. Los archivos generados se pueden integrar automáticamente a un sistema de control de versiones.
Estos son los caracteres codificados:

Carácter	Codificación
"	%22
*	%2A
/	%2F
:	%3A
<	%3C
>	%3E
?	%3F
|	%7C
\	%5C
%	%25

Ejemplos:
Form?1 es codificado *Form%3F1*
Button/1 es codificado *Button%2F1*

Current method path

Current method path -> Resultado

Parámetro	Tipo		Descripción
Resultado	Texto		Ruta interna completa del método en ejecución

Descripción

El comando **Current method path** devuelve la ruta de acceso interna del método base, trigger, método de proyecto, método formulario o método objeto en ejecución.

Nota: en el contexto de los macro comandos 4D, la etiqueta <*method_path*> es remplazada por la ruta de acceso completa del código en ejecución.

FORM GET NAMES

FORM GET NAMES ({tabla ;} arrayNoms {; filtro {; marcador}}{; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Referencia de tabla
arrayNoms	Array texto	⬅	Array de nombres de formulario
filtro	Texto	➡	Filtro de nombres
marcador	Entero largo	➡	Marcador para versión mínima a devolver
		⬅	Nuevo valor
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **FORM GET NAMES** llena el array *arrayNoms* con los nombres de los formularios de la aplicación.

Si pasa el parámetro *tabla*, el comando devuelve los nombres de los formularios tabla asociados a esta tabla. Si omite este parámetro, el comando devuelve los nombres de los formularios proyecto de la base.

Puede limitar esta lista de formularios pasando una cadena de comparación en el parámetro *filtro*: en este caso, sólo se devuelven los formularios cuyos nombres corresponden al filtro. Puede utilizar el carácter @ para definir los filtros de tipo "comienza por", "termina en" o "contiene". Si pasa una cadena vacía, se ignora el parámetro *filtro*.

También puede limitar la lista de formularios utilizando el parámetro opcional *marcador*, que permite limitar los formularios devueltos en *arrayNoms* a los que fueron modificados después de un tiempo determinado. Como parte de un sistema de control de versiones, este parámetro le permite actualizar sólo los formularios que se fueron modificados desde la última copia de seguridad.

Este principio funciona de esta forma: 4D mantiene un contador interno de las modificaciones de los recursos de la base. Como los formularios son recursos, cada vez que un formulario se crea o se guarda, el contador se incrementa.
Si pasa el parámetro *marcador*, el comando devuelve, en *arrayNoms*, sólo los formularios que corresponden a los valores de los marcadores superiores o iguales a *marcador*. Además, si pasa una variable en el marcador, el comando devuelve el nuevo valor del contador de modificación, es decir, el más alto, en esta variable. Puede guardar este valor y utilizarlo en la próxima llamada al comando **FORM GET NAMES** para recuperar sólo los formularios nuevos o modificados.

Si el comando se ejecuta desde un componente, devuelve por defecto los nombres de los formularios proyecto del componente. Si pasa el parámetro *, el array contendrá los formularios de la base local.

Nota: los formularios ubicados en la papelera no se listan.

Ejemplo

Ejemplos de uso típicos:

```
  // Lista de todos los formularios proyecto de la base
FORM GET NAMES(arr_Names)

  // Lista de formularios de la tabla [Empleados]
FORM GET NAMES([Empleados] ;arr_Names)

  // Lista de formularios "input" de la tabla [Empleados]
FORM GET NAMES([Empleados] ;arr_Names;"input_@")

  // Lista de formularios proyecto especificos de la base
FORM GET NAMES(arr_Names;"dialogue_@")</p><p>  // Lista todos los formularios proyecto que fueron modificados desde la última sincronización
  // vMarker contiene un valor numérico
FORM GET NAMES(arr_Names;"";vMarker)

  // Lista de formularios tabla desde un componente
  // Un puntero es necesario porque se desconoce el nombre de la tabla
FORM GET NAMES(tablePtr->;arr_Names;*)
```

METHOD Get attribute

METHOD Get attribute (ruta ; tipoAtributo {; *}) -> Resultado

Parámetro	Tipo		Descripción
ruta	Texto	➡	Ruta de método proyecto
tipoAtributo	Entero largo	➡	Tipo de atributo a obtener
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)
Resultado	Booleano	⮌	True = atributo seleccionado; de lo contrario False

Descripción

El comando **METHOD Get attribute** devuelve el valor del atributo *tipoAtributo* para el método proyecto designado por el parámetro *ruta*. Este comando sólo funciona con métodos proyecto. Si pasa una *ruta* invalida, se genera un error.

En el parámetro *tipoAtributo*, pase un valor indicando el tipo de atributo a obtener. Puede utilizar las siguientes constantes del tema **Acceso objetos diseño**:

Constante	Tipo	Valor	Comentario
Attribute executed on server	Entero largo	8	
Attribute invisible	Entero largo	1	Corresponde a la opción "Invisible"
Attribute published SOAP	Entero largo	3	Corresponde a la opción "Ofrecido como servicio web"
Attribute published SQL	Entero largo	7	Corresponde a la opción "Disponible vía SQL"
Attribute published Web	Entero largo	2	Corresponde a la opción "Disponible vía las etiquetas HTML y los URLs 4D (4DACTION...)"
Attribute published WSDL	Entero largo	4	Corresponde a la opción "Publicado en WSDL"
Attribute shared	Entero largo	5	Corresponde a la opción "Compartido entre componentes y base local"

Si el comando se ejecuta desde un componente, se aplica por defecto a los métodos del componente. Si pasa el parámetro *, accede a los métodos de la base local.

El comando devuelve **True** si se selecciona un atributo y **False** si se deselecciona.

METHOD GET ATTRIBUTES

METHOD GET ATTRIBUTES (ruta ; atributos {; *})

Parámetro	Tipo		Descripción
ruta	Texto, Array texto	➡	Rutas de métodos
atributos	Objeto, Array objeto	⬅	Atributos para los métodos seleccionados
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

Tema: Acceso objetos diseño

El nuevo comando **METHOD GET ATTRIBUTES** devuelve, en *atributos*, el valor actual de todos los atributos para los métodos especificados en el parámetro *ruta*.

Este comando sólo funciona con métodos proyecto. Si pasa una *ruta* no válida, se genera un error.

En *ruta*, puede pasar ya sea un texto que contenga una ruta de método, o un array texto que contenga un array de rutas. Debe pasar el mismo tipo de parámetro (variable o array) en *atributos* con el fin de obtener los atributos apropiados.

En *atributos*, pase un objeto o un array de objetos, dependiendo del tipo de parámetro pasado en *ruta*. Todos los atributos de métodos se devuelven como propiedades de objeto, con los valores "True"/"False" para los atributos Booleanos, los valores texto o los valores adicionales si es necesario (por ejemplo, "scope":"table" para la propiedad 4D Mobile).

Si el comando se ejecuta desde un componente, por defecto se aplica a los métodos del componente. Si pasa el parámetro *, accede a los métodos de la base local.

Nota: el comando existente **METHOD Get attribute** aún se soporta pero como sólo puede devolver valores booleanos, no puede ser utilizado para atributos extendidos, tales como las propiedades 4D Mobile.

Ejemplo

Usted quiere obtener los atributos del método de proyecto *sendMail*. Puede escribir:

```
C_OBJECT($att)
METHOD GET ATTRIBUTES("sendMail";$att)
```

Después de la ejecución, $att contiene, por ejemplo:

```
{ "invisible":false, "preemptive":"capable", "publishedWeb":false, "publishedSoap":false, "publishedWsdl":false, "shared":false,
"publishedSql":false, "executedOnServer":false, "published4DMobile":{ "scope":"table", "table":"Table_1" } }
```

METHOD GET CODE

METHOD GET CODE (ruta ; codigo {; opcion} {; *})

Parámetro	Tipo		Descripción
ruta	Texto, Array texto	➡	Texto o array de texto que contiene una o varias rutas de método
codigo	Texto, Array texto	⬅	Código de los métodos designados
opcion	Entero largo	➡	0 o si se omite = exportación simple (sin tokens), 1 = exportación con tokens
*	Operador	➡	Si se pasa = comando se aplica a la base de datos de host cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD GET CODE** devuelve en el parámetro *codigo*, el contenido de los métodos designados por el parámetro *ruta*. Este comando puede devolver el código de todos los tipos de métodos: métodos base, triggers, métodos proyecto, métodos formulario y métodos objeto.

Puede utilizar dos tipos de sintaxis, basadas en arrays texto o variables texto:

```
C_TEXT(tVpath)  // variables texto
C_TEXT(tVcode)
METHOD GET CODE(tVpath;tVcode)  // código de un solo método
```

```
ARRAY TEXT(arrPaths;0)  // arrays texto
ARRAY TEXT(arrCodes;0)
METHOD GET CODE(arrPaths;arrCodes)  // códigos de varios métodos
```

No se pueden combinar las dos sintaxis.

Si pasa una ruta de acceso no válida, el parámetro *codigo* se deja vacío y se genera un error.

En el texto del *codigo* generado por este comando:

- Los nombres de los comandos se escriben en inglés, excepto si utiliza una versión francesa de 4D y si tiene seleccionada la preferencia "Utilizar lenguaje francés y parámetros regionales sistema" (ver **Página Métodos**). Cuando se utiliza el parámetro *opcion*, el código puede contener tokens del lenguaje con el fin de que sea independiente del lenguaje de programación 4D y de la versión (ver más adelante).
- Para aumentar la legibilidad del código, el texto es indentado con los caracteres de tabulación en función de las estructuras de programación, al igual que en el editor de métodos.
- Una línea se añade en el encabezado del código generado que contiene los metadatos utilizados para la importación del código, por ejemplo:

```
// %attributes = {"lang":"fr","invisible":true,"folder":"Web3"}
```

  Durante una importación, esta línea no se importa, se utiliza para definir los atributos a aplicar (lo atributos no especificados se reinicializan a su valor por defecto). El atributo "lang" define el lenguaje de exportación e impide una importación en una aplicación en lenguaje diferente (en este caso, se genera un error). El atributo "folder" contiene el nombre de la carpeta padre del método; no se muestra cuando el método no tiene una carpeta padre.
  Pueden definirse atributos adicionales. Para mayor información, consulte la descripción del comando **METHOD SET ATTRIBUTES**.

El parámetro *opcion* le permite seleccionar el modo de exportación del código con respecto a los elementos del lenguaje tokenizados de los métodos:

- Si pasa 0 u omite el parámetro *opcion*, el código del método se exporta sin tokens, es decir, al igual que se muestra en el editor de métodos.
- Si pasa 1 o la constante Code with tokens, el código del método se exporta con tokens, es decir, los elementos tokenizados son seguidos por su referencia interna en el contenido del *código* exportado. Por ejemplo, la expresión "**String**(a)" se exporta "**String**:C10(a)", donde "C10" es la referencia interna del comando **String**.

Los elementos tokenizados del lenguaje incluyen:

- los comandos y constantes 4D,
- los nombres de tablas y campos,
- los comandos de plug-ins 4D.

El código exportado con sus tokens hace que sea independiente del lenguaje de programación 4D, y también de cualquier cambio de nombre posterior de los elementos del lenguaje. Gracias a los tokens, el código proporcionado en forma de texto siempre será

interpretado correctamente por 4D, por ejemplo utilizando el comando **METHOD SET CODE** o por copiar/pegar. Para más información sobre la sintaxis tokens 4D, consulte la sección **Utilizar tokens en fórmulas**.
Si el comando se ejecuta desde un componente, se aplica por defecto a los métodos del componente. Si pasa el parámetro *, accede a los métodos de la base local.

Ejemplo 1

Consulte el ejemplo del comando **METHOD SET CODE**.

Ejemplo 2

Este ejemplo ilustra el efecto del parámetro *opcion*.

Usted desea exportar el siguiente método "simple_init" :

```
Case of
	<p>:(Form event=On Load)
	ALL RECORDS([Customer])
End case
```

Si se ejecuta el siguiente código:

```
C_TEXT($code)
C_TEXT($contents)
$code:=METHOD Get path(Path project method;"simple_init")
METHOD GET CODE($code;$contents;0)  //sin tokens
TEXT TO DOCUMENT("simple_init.txt";$contents)
```

El documento resultante contendrá:

```
  //%attributes = {"lang":"fr"} comentario añadido y reservado por 4D
Case of
	: (Form event=On Load)
		ALL RECORDS([Customer])
End case
```

Si se ejecuta el siguiente código:

```
C_TEXT($code)
C_TEXT($contents)
$code:=METHOD Get path(Path project method;"simple_init")
METHOD GET CODE($code;$contents;Code with tokens)  //sin tokens
TEXT TO DOCUMENT("simple_init.txt";$contents)
```

El documento resultante contendrá:

```
  //%attributes = {"lang":"fr"} comentario añadido y reservado por 4D
Case of
	: (Form event:C388=On Load:K2:1)
		ALL RECORDS:C47([Customer:1])
End case
```

METHOD GET COMMENTS

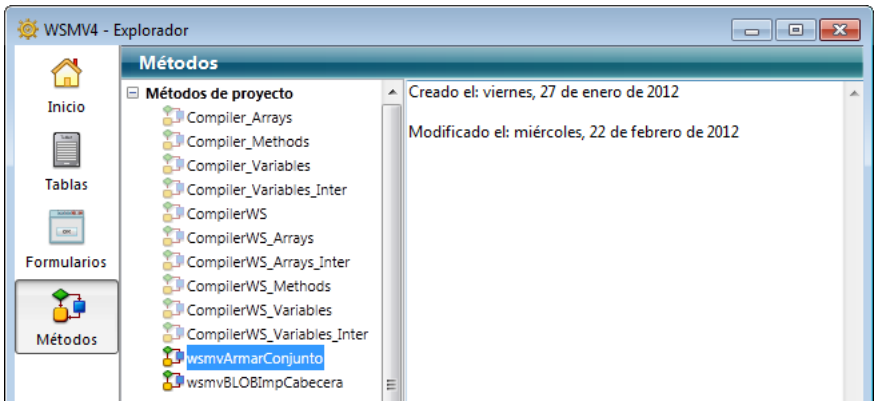
METHOD GET COMMENTS (ruta ; comentarios {; *})

Parámetro	Tipo		Descripción
ruta	Texto, Array texto	➡	Texto o array de texto que contiene una o varias rutas de método
comentarios	Texto, Array texto	⬅	Comentarios de los métodos designados
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD GET COMMENTS** devuelve en el parámetro *comentarios*, los comentarios de los métodos designados por el parámetro *ruta*.

Los comentarios recuperados por este comando son los especificados en el Explorador de 4D (no deben confundirse con las líneas de comentarios en el código que se recuperan utilizando **METHOD GET CODE**):



Estos comentarios sólo pueden ser generados para los métodos de tipo triggers, métodos proyecto y métodos formulario. Contienen texto con estilo.

Nota: los formularios y los métodos formulario comparten los mismos comentarios.
Puede utilizar dos tipos de sintaxis, basadas en arrays texto o en variables texto:

```
C_TEXT(tVpath)  // variables texto
C_TEXT(tVcomments)
METHOD GET COMMENTS(tVpath;tVcomments)  // comentarios de un solo método
```

```
ARRAY TEXT(arrPaths;0)  // arrays texto
ARRAY TEXT(arrComments;0)
METHOD GET COMMENTS(arrPaths;arrComments)  // comentarios de varios métodos
```

No se pueden combinar las dos sintaxis.

Si el comando se ejecuta desde un componente, se aplica por defecto a los métodos del componente. Si pasa el parámetro *, accede a los métodos de la base local.

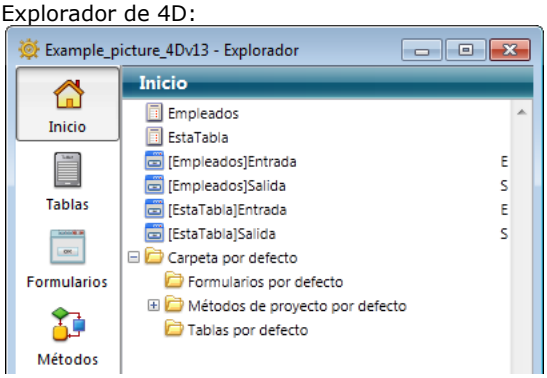
METHOD GET FOLDERS

METHOD GET FOLDERS (arrNoms {; filtro}{; *})

Parámetro	Tipo		Descripción
arrNoms	Array texto	⇐	Array de nombres de carpetas de la página de inicio
filtro	Texto	⇒	Filtro de nombres
*	Operador	⇒	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD GET FOLDERS** devuelve en el array *arrNoms*, los nombres de las carpetas creadas en la página Inicio del Explorador de 4D:



Como los nombres de las carpetas deben ser únicos, la jerarquía no se devuelve en este array.

Puede limitar la lista de carpetas pasando una cadena de comparación en el parámetro *filtro*, en este caso, sólo se devuelven las carpetas cuyos nombres coincidan con el filtro. Puede utilizar el carácter @ para definir los filtros de tipo "comienza por", "termina en" o "contiene". Si pasa una cadena vacía, el parámetro *filtro* se ignora.

Si se ejecuta este comando desde un componente, devuelve por defecto las rutas de los métodos del componente. Si pasa el parámetro *, el array contendrá las rutas de los métodos de la base local.

METHOD GET MODIFICATION DATE

METHOD GET MODIFICATION DATE (ruta ; fechaMod ; horaMod {; operador})

Parámetro	Tipo		Descripción
ruta	Texto, Array texto	➡	Texto o array texto que contiene una o más rutas de acceso
fechaMod	Fecha, Array fecha	⬅	Fecha(s) de modificación de métodos(s)
horaMod	Hora, Array entero largo	⬅	Hora(s) de modificación de métodos(s)
operador	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD GET MODIFICATION DATE** devuelve en los parámetros *fechaMod* y *horaMod* las fechas y horas de la última modificación de los métodos designados por el parámetro *ruta*.

Puede utilizar dos tipos de sintaxis, basadas en arrays o variables:

```
C_TEXT(tVpath)  // variables
C_DATE(vDate)
C_TIME(vTime)
METHOD GET MODIFICATION DATE(tVpath;vDate;vTime)  // fecha y hora de un solo método
```

```
ARRAY TEXT(arrPaths;0)  // arrays
ARRAY DATE(arrDates;0)
ARRAY LONGINT(arrTimes;0)
METHOD GET MODIFICATION DATE(arrPaths;arrDates;arrTimes)  // fechas y horas de varios métodos
```

No es posible combinar las dos sintaxis.

Si el comando se ejecuta desde un componente, se aplica por defecto a los métodos del componente. Si pasa el parámetro *, accede a los métodos de la base local.

Ejemplo 1

Quiere conocer las fechas y horas de modificación por varios métodos:

```
ARRAY TEXT(arrPaths;0)
APPEND TO ARRAY(arrPaths;"MyMethod1")
APPEND TO ARRAY(arrPaths;"MyMethod2")
...
ARRAY DATE(arrDates;0)
ARRAY LONGINT(arrTimes;0)
METHOD GET MODIFICATION DATE(arrPaths;arrDates;arrTimes)
```

Ejemplo 2

Quiere obtener las fechas de modificación de los métodos de un módulo con el prefijo "Web_". No se puede utilizar el símbolo "@" en una ruta; Sin embargo, puede escribir:

```
ARRAY TEXT($_webMethod;0)
METHOD GET NAMES($_webMethod;"Web_@")
ARRAY DATE($_date;0)
ARRAY LONGINT($_time;0)
METHOD GET MODIFICATION DATE($_webMethod;$_date;$_time)
```

METHOD GET NAMES

METHOD GET NAMES (arrNoms {; filtro}{; *})

Parámetro	Tipo		Descripción
arrNoms	Array texto	⇐	Array de nombres de métodos de proyecto
filtro	Texto	⇒	Filtros de nombres
*	Operador	⇒	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD GET NAMES** llena el array *arrNoms* con los nombres de los métodos proyecto creados en la aplicación.

Por defecto, todos los métodos se listan. Puede restringir esta lista pasando una cadena de comparación en el parámetro *filtro*, en este caso, el comando sólo devuelve los métodos cuyo nombre coincide con el filtro. Debe utilizar el carácter @ con el fin de definir los filtros de tipo "comienza por", "termina en" o "contiene".

Si se ejecuta este comando desde un componente, devuelve por defecto los nombres de los métodos proyecto del componente. Si pasa el parámetro *, el array contendrá los métodos proyecto de la base local.

Nota: los métodos ubicados en la papelera no se listan.

Ejemplo

Ejemplos de uso:

```
// Lista de todos los métodos proyecto de la base
METHOD GET NAMES(t_Names)

// Lista de los métodos proyecto que comienzan por una cadena específica
METHOD GET NAMES(t_Names;"web_@")

// Lista de los métodos proyecto de la base local que comienzan por una cadena específica
METHOD GET NAMES(t_Names;"web_@";*)
```

METHOD Get path

METHOD Get path (tipoMetodo {; laTabla}{; nomObjeto{; nomObjetoForm}}{; *}) -> resultado

Parámetro	Tipo		Descripción
tipoMetodo	Entero largo	➡	Selector de tipo de objeto
laTabla	Tabla	➡	Referencia de tabla
nomObjeto	Texto	➡	Nombre de formulario o método base
nomObjetoForm	Texto	⬅	Nombre del objeto de formulario
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)
resultado	Texto	↩	Ruta de acceso completa del objeto

Descripción

El comando **METHOD Get path** devuelve la ruta de acceso interna completa de un método.

Pase en *tipoMetodo*, el tipo de método cuya ruta quiere obtener. Puede utilizar las siguientes constantes, del tema **Acceso objetos diseño**:

Constante	Tipo	Valor	Comentario
Path database method	Entero largo	2	
Path project form	Entero largo	4	Ruta de los métodos formularo proyecto y de todos su métodos objeto. Ejemplos: *[projectForm]/myForm/{formMethod}* *[projectForm]/myForm/button1* *[projectForm]/myForm/my%2list* *[projectForm]/myForm/button1*
Path project method	Entero largo	1	Nombre del método. Ejemplo: MiMetodoProyecto
Path table form	Entero largo	16	Ruta de los métodos formulario tabla y de todos sus métodos objeto. Ejemplos: *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/button1* *[tableForm]/table_1/Form1/my%2list* *[tableForm]/table_2/Form1/my%2list*
Path trigger	Entero largo	8	Ruta de los triggers de la base. Ejemplos: *[trigger]/tabla_1* *[trigger]/tabla_2*

Pase los valores en los parámetros *laTabla*, *nomObjeto* y *nomObjetoForm* en función del tipo de objeto del cual quiere obtener la ruta de acceso del método:

Tipo de objeto	*laTabla*	*nomObjeto*	*nomObjetoForm*
Ruta Formulario proyecto		X	X (opcional)
Ruta Formulario tabla	X	X	X (opcional)
Ruta Método base		X	
Ruta Método proyecto		X	
Ruta Trigger	X		

Si el objeto no se encuentra (tipo de método desconocido o no valido, tabla faltante, etc.), se genera un error.

Si el comando se ejecuta desde un componente, devuelve por defecto las rutas de los métodos del componente. Si pasa el parámetro *, el array contendrá las rutas de los métodos de la base local.

Ejemplo

```
//Recuperar la ruta de acceso del método base "On Startup":
$path:=METHOD Get path(Path database method;"onStartup")

//Recuperar la ruta de acceso del trigger de la tabla [Empleados]:
$path:=METHOD Get path(Path trigger;[Empleados])

//Recuperar la ruta de acceso del método del objeto "OK" del formulario "input" de la tabla [Empleados]:
$path:=METHOD Get path(Path table form;[Empleados];"input";"OK")
```

METHOD GET PATHS

METHOD GET PATHS ({nomCarpeta ;} tipoMetodo ; arrRutas {; marcador}{; *})

Parámetro	Tipo		Descripción
nomCarpeta	Texto	➡	Nombre de carpeta de la página Inicio
tipoMetodo	Entero largo	➡	Selector de tipo de método a recuperar
arrRutas	Array texto	⬅	Array de rutas y nombres de los métodos
marcador	Variable entero largo	➡	Valor mínimo de marcador
		⬅	Nuevo valor actual
*	Operador	➡	Si se pasa = comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD GET PATHS** llena el array *arrRutas* con las rutas de acceso internas y los nombres de los métodos de la aplicación del tipo definido por el parámetro *tipoMetodo*.

Si su código está organizado en "carpetas" en el Explorador de 4D (página Inicio), puede pasar un nombre de carpeta en el parámetro opcional *nomCarpeta*. En este caso, el array *arrRutas* sólo contiene las rutas de los métodos ubicados en esta ubicación.

Nota: no puede utilizar el carácter comodín, "@" en *nomCarpeta*.

Pase en el parámetro *tipoMetodo* el tipo de método del cual quiere obtener la rutas en el array *arrRutas*. Puede utilizar las siguientes constantes (individualmente o en combinación), del tema **Acceso objetos diseño**:

Constante	Tipo	Valor	Comentario
Path all objects	Entero largo	31	Combinación de las rutas de todos los métodos de la base
Path database method	Entero largo	2	
Path project form	Entero largo	4	Ruta de los métodos formularo proyecto y de todos su métodos objeto. Ejemplos: *[projectForm]/myForm/{formMethod}* *[projectForm]/myForm/button1* *[projectForm]/myForm/my%2list* *[projectForm]/myForm/button1*
Path project method	Entero largo	1	Nombre del método. Ejemplo: MiMetodoProyecto
Path table form	Entero largo	16	Ruta de los métodos formulario tabla y de todos sus métodos objeto. Ejemplos: *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/button1* *[tableForm]/table_1/Form1/my%2list* *[tableForm]/table_2/Form1/my%2list*
Path trigger	Entero largo	8	Ruta de los triggers de la base. Ejemplos: *[trigger]/tabla_1* *[trigger]/tabla_2*

El parámetro *marcador* permite recuperar las rutas de los métodos modificados a partir de un momento específico. Como parte de un sistema de control de versión, esto significa que puede actualizar sólo los métodos modificados desde el último backup.
El funcionamiento es el siguiente: 4D mantiene un contador de modificación de métodos. Cada vez que un método se crea o se vuelve a guardar, este contador se incrementa y su valor actual se guarda en el marcador interno del método.
Si pasa el parámetro *marcador*, el comando sólo devuelve los métodos cuyo marcador es superior o igual al valor pasado en este parámetro. Además, el comando devuelve en *marcador* el nuevo valor actual del contador de modificación, es decir el valor más alto. Si guarda este valor, puede pasarlo la próxima vez que este comando se llame de manera que usted sólo recupera los métodos nuevos o modificados.

Si se ejecuta este comando desde un componente, devuelve por defecto las rutas de los métodos del componente. Si pasa el parámetro *, el array contendrá las rutas de los métodos de la base local.
Si el comando detecta un nombre de método duplicado, se genera el error -9802 ("Object path not unique"). En este caso, es aconsejable utilizar el CSM con el fin de verificar la estructura de la base de datos.

Ejemplo 1

Recuperación de los métodos de proyecto ubicados en una carpeta "web":

```
METHOD GET PATHS("web"; Path Project method;arrPaths)
```

Ejemplo 2

Recuperación de los métodos base y de los triggers:

```
METHOD GET PATHS(Path trigger+Path database method;arrPaths)
```

Ejemplo 3

Recuperación de los métodos de proyecto modificados desde el último backup:

```
 // cargamos el último valor almacenado
$stamp :=Max([Backups]cur_stamp)
METHOD GET PATHS(Path project method;arrPaths;$stamp)
 // guardamos el nuevo valor
CREATE RECORD([Backups])
[Backups]cur_stamp :=$stamp
SAVE RECORD([Backups])
```

Ejemplo 4

Consulte el ejemplo del comando **METHOD SET CODE**.

METHOD GET PATHS FORM

METHOD GET PATHS FORM ({laTabla ;} arrRutas {; filtro}{; marcador}{; *})

Parámetro	Tipo		Descripción
laTabla	Tabla	➡	Referencia de tabla
arrRutas	Array texto	⬅	Array de rutas y nombres de los métodos
filtro	Texto	➡	Filtros de nombres
marcador	Variable entero largo	➡	Valor mínimo de marcador
		⬅	Nuevo valor actual
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD GET PATHS FORM** llena el array *arrRutas* con las rutas de acceso internas y los nombres de los métodos de todos los objetos de los formularios así como también de los métodos formulario. Los métodos formulario se etiquetan {formMethod}.

Sólo los objetos que contienen código se listan. Por ejemplo, no se devuelven los botones que estén asociados únicamente con una acción estándar.

Si pasa el parámetro *laTabla*, el comando devuelve los objetos de los formularios tabla asociados a esta tabla. Si omite este parámetro, el comando devuelve los objetos de formularios proyecto de la base.

Puede limitar esta lista de formularios pasando una cadena de comparación en el parámetro *filtro*, en este caso, sólo se devuelven los formularios cuyos nombres coincidan con el filtro. Puede utilizar el carácter @ para definir los filtros de tipo "comienza por", "termina en" o "contiene". Si pasa una cadena vacía, el parámetro *filtro* se ignora.

El parámetro *marcador* permite recuperar las rutas de los métodos modificados a partir de un momento específico. Como parte de un sistema de control de versión, esto significa que puede actualizar sólo los métodos modificados desde el último backup. El funcionamiento es el siguiente: 4D mantiene un contador de modificación de métodos. Cada vez que un método se crea o se vuelve a guardar, este contador se incrementa y su valor actual se guarda en el marcador interno del método.
Si pasa el parámetro *marcador*, el comando sólo devuelve los métodos cuyo marcador es superior o igual al valor pasado en este parámetro. Además, el comando devuelve en *marcador* el nuevo valor actual del contador de modificación, es decir el valor más alto. Si guarda este valor, puede pasarlo la próxima vez que este comando se llame de manera usted sólo recupera los métodos nuevos o modificados.

Si se ejecuta este comando desde un componente, devuelve por defecto las rutas de los métodos del componente. Si pasa el parámetro *, el array contendrá las rutas de los métodos de la base local.

Nota: el comando no lista los objetos de los formularios heredados o de los subformularios.
Si el comando detecta un nombre de método duplicado, se genera el error -9802 ("Object path not unique"). En este caso, es aconsejable utilizar el CSM con el fin de verificar la estructura de la base de datos.

Ejemplo 1

Lista de todos los objetos del formulario "input" de la tabla [Empleados]. Note que los métodos formulario tabla (y los métodos formulario proyecto) se procesan como objetos que pertenecen al formulario:

```
METHOD GET PATHS FORM([Employees];arrPaths;"input")
  // Contenido de arrPaths (por ejemplo)
  // [tableForm]/input/{formMethod} -> Método formulario
  // [tableForm]/input/bOK -> Método objeto
  // [tableForm]/input/bCancel -> Método objeto
```

Ejemplo 2

Lista de los objetos del formulario proyecto "dial":

```
METHOD GET PATHS FORM(arrPaths;"dial")
```

Ejemplo 3

Lista de todos los objetos de los formularios "input" de la tabla [Empleados] a partir de un componente:

```
METHOD GET PATHS FORM(([Empleados];arrPaths;"input@";*)
```

METHOD OPEN PATH

METHOD OPEN PATH (ruta {; *})

Parámetro	Tipo		Descripción
ruta	Texto	➡	Ruta del método a abrir
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente(parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD OPEN PATH** abre, en el editor de métodos de 4D, el método cuya ruta de acceso interna se pasa en el parámetro *ruta*.

Este comando puede abrir todo tipo de métodos (objeto, formulario, trigger, proyecto o base); sin embargo. el método ya debe existir. Si el parámetro *ruta* no corresponde a un método existente, se devuelve el error -9801 "Imposible abrir el método".

Puede ejecutar este comando desde un componente, pero en este caso, debe pasar el parámetro * ya que el acceso al código del componente es de sólo lectura. Si omite el parámetro * en este contexto, se genera el error -9763.

METHOD RESOLVE PATH

METHOD RESOLVE PATH (ruta ; tipoMetodo ; prtTabla ; nomObjeto ; nomObjForm {; *})

Parámetro	Tipo		Descripción
ruta	Texto	➡	Ruta a resolver
tipoMetodo	Entero largo	⬅	Selector de tipo de objeto
prtTabla	Puntero	⬅	Referencia de tabla
nomObjeto	Texto	⬅	Nombre de formulario o de método base
nomObjForm	Texto	⬅	Nombre de objeto del formulario
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD RESOLVE PATH** analiza la ruta de acceso interna pasada en el parámetro *ruta* y devuelve sus diferentes componentes en los parámetros *tipoMetodo*, *ptrTabla*, *nomObjeto* y *nomObjetoForm*.

En el parámetro *tipoMetodo* devuelve un valor que indica el tipo del método. Puede comparar este valor con las siguientes constantes del tema **Acceso objetos diseño**:

Constante	Tipo	Valor	Comentario
Path database method	Entero largo	2	
Path project form	Entero largo	4	Ruta de los métodos formularo proyecto y de todos su métodos objeto. Ejemplos: *[projectForm]/myForm/{formMethod}* *[projectForm]/myForm/button1* *[projectForm]/myForm/my%2list* *[projectForm]/myForm/button1*
Path project method	Entero largo	1	Nombre del método. Ejemplo: MiMetodoProyecto
Path table form	Entero largo	16	Ruta de los métodos formulario tabla y de todos sus métodos objeto. Ejemplos: *[tableForm]/table_1/Form1/{formMethod}* *[tableForm]/table_1/Form1/button1* *[tableForm]/table_1/Form1/my%2list* *[tableForm]/table_2/Form1/my%2list*
Path trigger	Entero largo	8	Ruta de los triggers de la base. Ejemplos: *[trigger]/tabla_1* *[trigger]/tabla_2*

El parámetro *ptrTabla* contiene un puntero a una tabla de la base cuando la ruta referencia un método formulario tabla o un trigger.

El parámetro *nomObjeto* contiene:

- un nombre de formulario si la ruta referencia un formulario tabla o proyecto.
- un nombre de método base si la ruta referencia un método base.

El parámetro *nomObjetoForm* contiene un nombre de objeto de formulario si la ruta referencia un método objeto.

Si el comando se ejecuta desde un componente, considera por defecto que *ruta* designa un método del componente. Si pasa el parámetro *, considera que *ruta* designa un método de la base local.

Ejemplo 1

Resolución de una ruta de método base:

```
C_LONGINT($methodType)
C_POINTER($tablePtr)
C_TEXT($objectName)
C_TEXT($formObjectName)

METHOD RESOLVE PATH("[databaseMethod]/onStartup";$methodType;$tablePtr;$objectName;$formObjectName)
 // $methodType: 2
 // $tablePtr: Nil pointer
 // $objectName: "onStartup"
 // $formObjectName: ""
```

Ejemplo 2

Resolución de una ruta de objeto de método formulario tabla:

```
C_LONGINT($methodType)
C_POINTER($tablePtr)
C_TEXT($objectName)
C_TEXT($formObjectName)

METHOD RESOLVE PATH("[tableForm]/Table1/output%2A1/myVar%2A1";$methodType;$tablePtr;$objectName;$formObjectName)
 // $methodType: 16
 // $tablePtr: -> [Table1]
 // $objectName: "output*1"
 // $formObjectName: "Btn*1"
```

METHOD SET ACCESS MODE

METHOD SET ACCESS MODE (modo)

Parámetro	Tipo		Descripción
modo	Entero largo	➡	Modo de acceso a los objetos bloqueados

Descripción

El comando **METHOD SET ACCESS MODE** permite definir el comportamiento de 4D cuando intenta acceder en escritura a un objeto ya cargado en modificación por otro usuario o proceso. El alcance de este comando es la sesión actual.

En *modo*, pase una de las siguientes constantes del tema **Acceso objetos diseño**:

Constante	Tipo	Valor	Comentario
On object locked abort	Entero largo	0	La carga del objeto se aborta (funcionamiento por defecto)
On object locked confirm	Entero largo	2	4D muestra una caja de diálogo permitiéndole elegir entre intentar nuevamente o abortar. En modo remoto, esta opción no es soportada (la carga se abandona)
On object locked retry	Entero largo	1	4D intenta cargar el objeto hasta que sea liberado

METHOD SET ATTRIBUTE

METHOD SET ATTRIBUTE (ruta ; tipoAtrib ; valorAtrib {; tipoAtrib2 ; valorAtrib2 ; ... ; tipoAtribN ; valorAtribN}{; operador})

Parámetro	Tipo		Descripción
ruta	Texto	➡	Ruta del método proyecto
tipoAtrib	Entero largo	➡	Tipo de atributo
valorAtrib	Booleano, Texto	➡	True = seleccionar el atributo False = deseleccionar el atributo
operador	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD SET ATTRIBUTE** permite definir el valor del atributo *tipoAtrib* para el método proyecto designado por el parámetro *ruta*. Este comando sólo funciona con métodos proyecto. Si pasa una *ruta* invalida, se genera un error.

En el parámetro *tipoAtrib*, pase un valor que indique el tipo de atributo a definir. Puede utilizar las siguientes constantes, del tema **Acceso objetos diseño**:

Constante	Tipo	Valor	Comentario
Attribute executed on server	Entero largo	8	
Attribute folder name	Entero largo	1024	Nombre de la carpeta para el método (atributo "carpeta"). Cuando pase esta constante, debe pasar un nombre de carpeta en *attribValue*: • si el nombre corresponde a una carpeta válida, el método se coloca en esta carpeta padre, • si la carpeta no existe, el comando no cambia nada en el nivel de la carpeta padre, • si pasa una cadena vacía, el método se ubica al nivel de la raíz.
Attribute invisible	Entero largo	1	Corresponde a la opción "Invisible"
Attribute published SOAP	Entero largo	3	Corresponde a la opción "Ofrecido como servicio web"
Attribute published SQL	Entero largo	7	Corresponde a la opción "Disponible vía SQL"
Attribute published Web	Entero largo	2	Corresponde a la opción "Disponible vía las etiquetas HTML y los URLs 4D (4DACTION...)"
Attribute published WSDL	Entero largo	4	Corresponde a la opción "Publicado en WSDL"
Attribute shared	Entero largo	5	Corresponde a la opción "Compartido entre componentes y base local"

Puede pasar en el parámetro *valorAtrib:*

- **True** para seleccionar la opción correspondiente y **False** para deseleccionarla o,
- una cadena (nombre de carpeta) si utilizó la constante Attribute folder name en *tipoAtrib*.

Puede pasar múltiples pares *tipoAtrib*;*valorAtrib* en una sola llamada.

Puede ejecutar este comando desde un componente, pero en este caso debe pasar el parámetro *, por el acceso en sólo escritura al código del componente. Si omite el parámetro * en este contexto, se genera el error -9763.

Este comando no puede ejecutarse en modo compilado. Cuando se llama en este modo, generará el error -9762.

Ejemplo 1

Selección de la propiedad "Compartido entre componentes y la base local" para el método proyecto "Selección diálogo":

```
METHOD SET ATTRIBUTE("Selección diálogo";Attribute shared;True)
```

Ejemplo 2

Definición de varios pares de atributos/valores:

```
METHOD SET ATTRIBUTE(vPath;Attribute invisible;vInvisible;Attribute published Web;v4DAction;Attribute published SOAP;vSoap;Attribute published WSDL;vWSDL;Attribute shared;vExported;Attribute published SQL;vSQL;Attribute executed on server;vRemote;Attribute folder name;vFolder;*)
```

METHOD SET ATTRIBUTES

METHOD SET ATTRIBUTES (ruta ; atributos {; *})

Parámetro	Tipo		Descripción
ruta	Texto, Array texto	➡	Rutas de métodos
atributos	Objeto, Array objeto	➡	Atributos para definir los métodos seleccionados
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

Tema: Acceso objetos diseño

El nuevo comando **METHOD SET ATTRIBUTES** le permite definir los valores de los *atributos* para lo métodos especificados en el parámetro *ruta*.

En *ruta*, puede pasar ya sea un texto que contiene una ruta de método, o un array de texto que contiene una gran variedad de rutas. Debe pasar el mismo tipo de parámetro (cadena o array) en *atributos* con el fin de establecer los atributos adecuados. Este comando sólo funciona con métodos proyecto. Si pasa una *ruta*, no válida, se genera un error.

En *atributos*, se pasa un objeto o un array de objetos (en función del tipo de parámetro pasado en *ruta*) que contiene todos los atributos que desea definir para los métodos.

Los atributos de métodos deben definirse con los comandos **OB SET** o **OB SET ARRAY**, con los valores True o False para los atributos booleanos, o los valores específicos para los atributos extendidos (por ejemplo, "scope":"table" para la propiedad 4D Mobile). Sólo los atributos que están presentes en el parámetro *atributos* se actualizarán en los atributos de los métodos.

Si el comando se ejecuta desde un componente, por defecto se aplica a los métodos del componente. Si pasa el parámetro *, accede a los métodos de la base local.

Nota: el comando existente **METHOD SET ATTRIBUTE** aún se soporta pero ya que sólo puede manejar valores booleanos, no se puede utilizar para atributos extendidos tales como propiedades 4D Mobile.

Los atributos soportados son:

{ "invisible" : false, // true si visible "preemptive" : "capable" // o "incapable" o "indifferent" "publishedWeb" : false, // true si está disponible a través de las etiquetas y URLs 4D "publishedSoap": false, // true si se ofrece como servicio web "publishedWsdl": false, // true si se ha publicado en WSDL "shared" : false, // true si es compartida por los componentes y la base local "publishedSql" : false, // true si está disponible a través de SQL "executedOnServer" : false, // true si se ejecuta en el servidor "published4DMobile" : { "scope": "table", // "none" o "table" o "currentRecord" o "currentSelection" "table": "aTableName" // presente si el alcance es diferente de "none" } }

Nota: para los atributos "published4DMobile", si el valor "table" no existe o si el "scope" no es válido, estos atributos se ignoran.

Ejemplo 1

Usted desea modificar un solo atributo:

```
C_OBJECT($attributes)
OB SET($attributes;"executedOnServer";True)
METHOD SET ATTRIBUTES("aMethod";$attributes)  //solo el atributo "executedOnServer" se modifica
```

Ejemplo 2

Usted desea que un método no esté disponible para 4D Mobile (el valor "none" debe pasarse para el atributo "scope"):

```
C_OBJECT($attributes)
C_OBJECT($fourDMobileAttribute)
OB SET($fourDMobileAttribute;"scope";"none")
OB SET($attributes;"published4DMobile";$fourDMobileAttribute)
METHOD SET ATTRIBUTES("aMethod";$attributes)
```

METHOD SET CODE

METHOD SET CODE (ruta ; codigo {; operador})

Parámetro	Tipo		Descripción
ruta	Texto, Array texto	➡	Texto o array texto que contiene una o varias rutas de métodos
codigo	Texto, Array texto	➡	Código de los métodos designados
operador	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD SET CODE** modifica el código de los métodos designados por el parámetro *ruta* con el contenido pasado en el parámetro *codigo*. Este comando puede acceder al código de todos los tipos de métodos: métodos base, triggers, métodos proyecto, métodos formulario y métodos objeto.

En el caso de un método proyecto: si este método ya existe en la base, su contenido se remplaza; si no existe, se crea con sus contenidos.

Puede utilizar dos tipos de sintaxis, basadas en arrays texto o en variables texto:

```
C_TEXT(tVpath)  // variables texto
C_TEXT(tVcode)
METHOD SET CODE(tVpath;tVcode)  // código de un sólo método
```

```
ARRAY TEXT(arrPaths;0)  // arrays texto
ARRAY TEXT(arrCodes;0)
METHOD SET CODE(arrPaths;arrCodes)  // código de varios métodos
```

No es posible mezclar las dos sintaxis.

Si pasa una ruta de acceso invalida, el comando no hace nada.

Cuando se llama METHOD SET CODE, los atributos de los métodos se reinicializan por defecto. Sin embargo, si la primera línea del *codigo* de un método contiene metadatos validos (expresados en JSON), se utilizan para definir los atributos del método y no se inserta la primera línea. Ejemplo de metadatos:

```
// %attributes = {"invisible":true,"lang":"fr","folder":"Security"}
```

Nota: estos metadatos son generados automáticamente por el comando **METHOD GET CODE**. Para más información sobre los atributos soportados, consulte la descripción del comando **METHOD SET ATTRIBUTES**.

Concerniente a la propiedad "folder" de los metadatos:

- si esta propiedad está presente y corresponde a una carpeta válida, el método se ubica en su carpeta padre,
- si esta propiedad no está presente o si la carpeta ya no existe, el comando no cambia a nivel de la carpeta padre,

Puede ejecutar el comando desde un componente, en este caso, debe pasar el parámetro * por el acceso al código del componente en modo sólo lectura. Si omite el parámetro * en este contexto, se genera el error -9763l.

Ejemplo

Este ejemplo exporta e importa la totalidad de los métodos proyecto de una aplicación:

```
$root_t:=Get 4D folder(Database folder)+"methods"+Folder separator
ARRAY TEXT($fileNames_at;0)
CONFIRM("Import or export methods?";"Import";"Export")

If(OK=1)
   DOCUMENT LIST($root_t;$fileNames_at)
   For($loop_l;1;Size of array($fileNames_at))
      $filename_t:=$fileNames_at{$loop_l}
      DOCUMENT TO BLOB($root_t+$filename_t;$blob_x)
      METHOD SET CODE($filename_t;BLOB to text($blob_x;UTF8 text without length))
   End for
Else
   If(Test path name($root_t)#Is a folder)
      CREATE FOLDER($root_t;*)
   End if
   METHOD GET PATHS(Path project method;$fileNames_at)
   METHOD GET CODE($fileNames_at;$code_at)
   For($loop_l;1;Size of array($fileNames_at))
```

```
        $filename_t:=$fileNames_at{$loop_l}
        SET BLOB SIZE($blob_x;0)
        TEXT TO BLOB($code_at{$loop_l};$blob_x;UTF8 text without length)
        BLOB TO DOCUMENT($root_t+$filename_t;$blob_x)
    End for
End if
SHOW ON DISK($root_t)
```

METHOD SET COMMENTS

METHOD SET COMMENTS (ruta ; comentarios {; Operador})

Parámetro	Tipo		Descripción
ruta	Texto, Array texto	➡	Texto o array texto que contiene una o varias rutas de métodos
comentarios	Texto, Array texto	➡	Comentarios de los métodos designados
Operador	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

El comando **METHOD SET COMMENTS** remplaza los comentarios de los métodos designados por el parámetro *ruta* por los definidos en el parámetro *comentarios*.

Los comentarios modificados por este comando son los definidos en el Explorador de 4D (no confundir con las líneas de comentarios en el código). Estos comentarios sólo pueden ser generados por los métodos de tipo triggers, métodos proyecto y métodos formulario. Contienen texto con estilo.

Nota: los formularios y los métodos formulario comparten los mismos comentarios.

Puede utilizar dos tipos de sintaxis, basadas en arrays texto o variables texto:

```
C_TEXT(tVpath)  // variables texto
C_TEXT(tVcomments)
METHOD SET COMMENTS(tVpath;tVcomments)  // comentarios de un solo método
```

```
ARRAY TEXT(arrPaths;0)  // arrays texto
ARRAY TEXT(arrComments;0)
METHOD SET COMMENTS(arrPaths;arrComments)  // comentarios para varios métodos
```

No es posible combinar las dos sintaxis.

Si pasa un nombre de ruta invalido, se genera un error.

Puede ejecutar este comando desde un componente, pero en este caso debe pasar el parámetro * porque el acceso en modo escritura al código del componente no es posible. Si omite el parámetro * en este contexto, se genera el error -9763.

Ejemplo

Añadir una fecha de modificación a un comentario de trigger existente:

```
METHOD GET COMMENTS("[trigger]/Table1";$comments)
$comments:="Modif:"+String(Current date)+"\r"+$comments
METHOD SET COMMENTS("[trigger]/Table1";$comments)
```

Área web

- Gestión programada de áreas web
- WA Back URL available
- WA Create URL history menu
- WA Evaluate JavaScript
- WA EXECUTE JAVASCRIPT FUNCTION
- WA Forward URL available
- WA Get current URL
- WA GET EXTERNAL LINKS FILTERS
- WA Get last filtered URL
- WA GET LAST URL ERROR
- WA Get page content
- WA Get page title
- WA GET PREFERENCE
- WA GET URL FILTERS
- WA GET URL HISTORY
- WA OPEN BACK URL
- WA OPEN FORWARD URL
- WA OPEN URL
- WA REFRESH CURRENT URL
- WA SET EXTERNAL LINKS FILTERS
- WA SET PAGE CONTENT
- WA SET PAGE TEXT LARGER
- WA SET PAGE TEXT SMALLER
- WA SET PREFERENCE
- WA SET URL FILTERS
- WA STOP LOADING URL

Gestión programada de áreas web

Los comandos de este tema están dedicados a la gestión programada de objetos de formulario de tipo área web.

Las áreas web pueden mostrar todo tipo de contenido web (*) al interior mismo de su entorno 4D: páginas HTML con contenidos estáticos o dinámicos, archivos, imágenes, Javascript. La siguiente imagen muestra un área web incluida en un formulario y muestra una página HTML:

(*) Sin embargo, no se recomienda el uso de plugins Web y applets Java (ver **Notas sobre uso de áreas web**).

Además de los comandos del tema web Area, varias acciones y eventos de formularios dedicados permiten al desarrollador controlar el funcionamiento de las áreas web. Pueden utilizarse variables específicas para intercambiar información entre el área y el entorno 4D. Estas herramientas pueden utilizarse para desarrollar un navegador web básico en sus formularios.

Crear y direccionar un área web

La creación de un área web se efectúa con la ayuda de una variante del botón Area de plug-in/Sub formulario de la barra de objetos del editor de formularios de 4D (para mayor información, consulte la sección **Áreas web** en el manual de *Diseño*).

Nota: cuando se muestra un área web que utiliza el motor el motor de renderización integrado en nuevo proceso creado con el comando **New process**, es necesario utilizar el valor por defecto (0) para el parámetro *pila*.

Como los otros objetos dinámicos del formulario, un área web dispone de un nombre de objeto y de un nombre de variable, que pueden ser utilizados para manejarla por programación. La variable estándar asociada al objeto área web es de tipo Texto. Específicamente, puede utilizar los comandos **OBJECT SET VISIBLE** y **OBJECT MOVE** con las áreas web.

Nota: la variable Texto asociada al área web no contiene una referencia por lo tanto no puede pasarse como parámetro a un método. Por ejemplo, para un área web llamada **MiArea**, no puede utilizarse el siguiente código:

```
Mymethod(MyArea)
```

Código para Mymethod:

```
WA REFRESH CURRENT URL($1) //No funciona
```

Para este tipo de programación, necesitará utilizar punteros:

```
Mymethod(->MyArea)
```

Código para Mymethod:

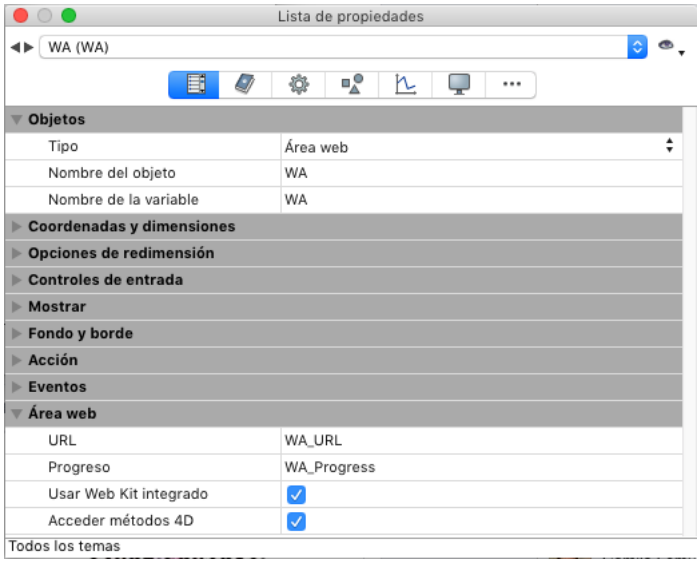
```
WA REFRESH CURRENT URL($1->) //Funciona
```

Gestión de variables asociadas

Además de las variables de objetos estándar (ver párrafo anterior), dos variables específicas son asociadas automáticamente a cada área web:

- La variable "URL"
- La variable "Progression".

Puede nombrar estas variables como prefiera. Estas variables son accesibles en la Lista de propiedades:



Variable URL

La variable "URL" es de tipo cadena. Contiene la URL cargada o que está siendo cargada por el área web asociada.

La asociación entre la variable y el área web funciona en ambas direcciones:

- Si el usuario asigna un nuevo URL a la variable, esta URL es cargada automáticamente por el área web.
- Toda navegación efectuada al interior del área web actualizará automáticamente el contenido de la variable.

Esquemáticamente, esta variable funciona como el área de dirección de un navegador web. Puede representarla por un área de texto situada sobre el área Web.

Variable URL y comando WA OPEN URL

La variable URL produce los mismos efectos que el comando **WA OPEN URL**. Sin embargo se deben tener en cuenta las siguientes diferencias:

- para acceder a los documentos, esta variable sólo acepta los URLs conformes a los RFC ("file://c:/Mi%20Doc") y no las rutas de acceso sistema ("c:\MiDoc"). El comando **WA OPEN URL** acepta las dos notaciones.
- si la variable URL contiene una cadena vacía, el área web no intenta cargar el URL. El comando **WA OPEN URL** genera un error en este caso.
- si la variable URL no contiene un protocolo (http, mailto, file, etc.), el área web añade "http://", que no es el caso para el comando **WA OPEN URL**.
- cuando el área web no se muestra en el formulario (cuando se ubica en otra página del formulario), la ejecución del comando **WA OPEN URL** no tiene efecto, mientras que la asignación de un valor a la variable URL puede utilizarse para actualizar el URL actual.

Variable Progreso

"Progreso" es una variable de tipo Entero largo. Contiene un valor entre 0 y 100, representa el porcentaje de carga completo de la página mostrada en el área web.

Esta variable es actualizada automáticamente por 4D. No es posible modificarla manualmente.

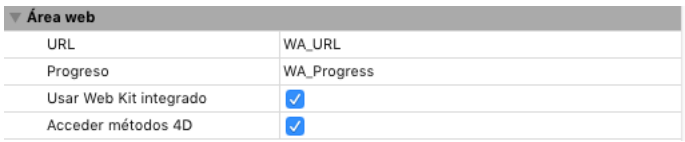
Acceder a los métodos 4D

Puede llamar a los métodos 4D desde el código JavaScript ejecutado en un área web y obtener valores a cambio.

Importante: esta funcionalidad sólo está disponible si el área web utiliza el motor de renderización Web integrado.

Configurar el área web

Para poder llamar a los métodos 4D desde un área web, debe seleccionar la opción **Acceso métodos 4D** para el área en la Lista de propiedades:



Nota: esta opción sólo se muestra cuando se selecciona la opción **Utilizar el motor de renderización Web integrado**.

Cuando esta propiedad está seleccionada, un objeto JavaScript especial ($4d) se instancia en el área web y permite gestionar las llamadas a los métodos proyecto 4D.

Uso del objeto $4d

Cuando la opción **Acceso a métodos 4D está seleccionada**, el motor de renderización Web integrado de 4D da al área un objeto JavaScript llamado *$4d* que se puede asociar con cualquier método proyecto 4D utilizando la notación objeto ".".

Por ejemplo, para llamar al método 4D ***HelloWorld***, sólo ejecute la siguiente instrucción:

```
$4d.HelloWorld();
```

Atención: JavaScript distingue entre mayúsculas y minúsculas, por lo que es importante tener en cuenta que el objeto se llama $4d (con "d" minúscula).

La sintaxis de las llamadas a métodos 4D es la siguiente:

```
$4d.4DMethodName(param1,paramN,function(result,error){})
```

- *param1...paramN*: puede pasar tantos parámetros como sea necesario al método 4D.
  Estos parámetros pueden ser de cualquier tipo soportado por JavaScript (cadena, número, array, objeto).
- *function(result)*: función a pasar como último argumento. Esta función de "retrollamada" se llama sincrónicamente una vez que el método 4D termina de ejecutarse. Recibe el parámetro *result.*
  - *result*: resultado de la ejecución del método 4D, devuelto en la expresión "$0".
    Este resultado puede ser de cualquier tipo compatible con JavaScript (cadena, número, array, objeto). Puede utilizar el comando **C_OBJECT** para devolver los objetos.
    Nota: Por defecto, 4D trabaja en UTF-8. Cuando regrese texto que contiene caracteres extendidos, por ejemplo, los caracteres con tilde, asegúrese de que la codificación de la página visualizada en el área web se declare como UTF-8, de lo contrario los caracteres pueden ser renderizados de forma incorrecta. En este caso, añada la siguiente línea en la página HTML para declarar la codificación:
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

Ejemplo 1

Dado un método proyecto 4D llamado ***today*** que no recibe parámetros y devuelve la fecha actual como una cadena.

Código 4D del método ***today***:

```
C_TEXT($0)
$0:=String(Current date;System date long)
```

En el área web, el método 4D puede ser llamado con la siguiente sintaxis:

```
$4d.today()
```

El método 4D no recibe ningún parámetro pero devuelve el valor de $0 a la función de retrollamada llamada por 4D después de la ejecución del método.
Queremos mostrar la fecha en la página HTML que es cargada por el área web.

Aquí está el código de la página HTML:

```
<html> <head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /> <script type="text/javascript">
$4d.today(function(dollarZero) {       var curDate = dollarZero;       document.getElementById("mydiv").innerHTML=curDate; }); </script>
</head> <body>Today is: <div id="mydiv"></div> </body> </html>
```

Ejemplo 2

El método proyecto 4D ***calcSum*** recibe los parámetros ($1...$n) y devuelve su suma de $0:
Código 4D del método ***calcSum***:

```
C_REAL(${1}) // recibes en los parámetros de tipo REAL
C_REAL($0) // devuelve un Real
C_LONGINT($i;$n)
$n:=Count parameters
For($i;1;$n)
   $0:=$0+${$i}
End for
```

El código JavaScript que ejecutará en el área web es:

```
$4d.calcSum(33, 45, 75, 102.5, 7, function(dollarZero)       {            var result = dollarZero // resultado es 262.5       });
```

Eventos formulario

Los eventos formulario específicos están destinados a la gestión programada de áreas web, particularmente a la activación de enlaces:

- **On Begin URL Loading**
- **On URL Resource Loading**
- **On End URL Loading**
- **On URL Loading Error**
- **On URL Filtering**
- **On Open External Link**
- **On Window Opening Denied**

Además, las áreas web soportan los siguientes eventos formulario genéricos:

- **On Load**
- **On Unload**
- **On Getting Focus**
- **On Losing Focus**

Para mayor información sobre estos eventos, consulte la descripción del comando **Form event**.

Acceso al inspector web

Puede ver y utilizar un inspector web dentro de las áreas web de sus formularios. El inspector web es un depurador, suministrado por el motor de renderización Web integrado, que analiza el código y el flujo de información de las páginas web.

Mostrar el inspector web

Las siguientes condiciones se deben cumplir para poder ver el inspector web en un área web:

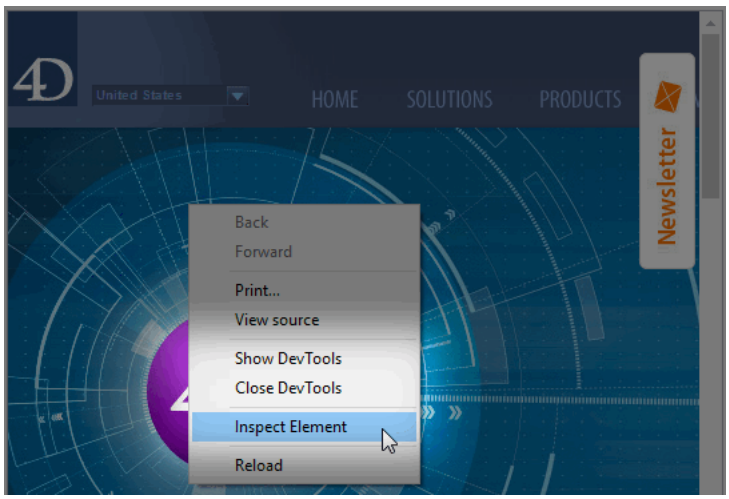
- Debe seleccionar el motor de renderización Web integrado para el área (el inspector web no está disponible en esta configuración) (ver **Uso del motor de renderización Web integrado**)
- Debe habilitar el menú contextual del área (este menú se utiliza para llamar al inspector) (ver **Menú contextual**)
- Debe habilitar expresamente el uso del inspector en el área por medio de la siguiente instrucción:

```
WA SET PREFERENCE(*;"WA";WA enable Web inspector;True)
```

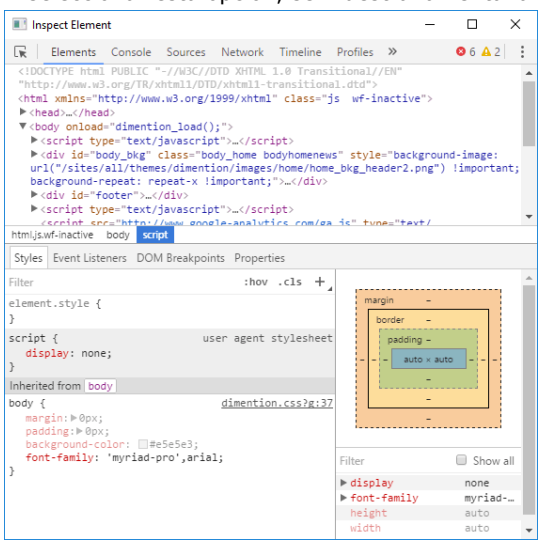
Para más información, consulte la descripción del comando **WA SET PREFERENCE**.

Uso del inspector web

Cuando haya hecho los ajustes como se describió arriba, tendrá nuevas opciones tales como **Inspeccionar Elemento** en el menú contextual del área:



Al seleccionar esta opción, se muestra la ventana del inspector Web.



El inspector web está incluido en el motor de renderización Web integrado. Para una descripción detallada de las funcionalidades de este depurador, consulte la documentación ofrecida por el motor de renderización web.

Notas sobre uso de áreas web

Interfaz usuario

Durante la ejecución del formulario, el usuario dispone de las funciones de interfaz estándar de los navegadores en el área web, lo cual permite la interacción con las otras áreas del formulario:

- **Comandos del menú Edición:** cuando el área web tiene el foco, los comandos del menú **Edición** permiten efectuar acciones como copiar, pegar, seleccionar todo, etc., de acuerdo a la selección.
- **Menú contextual:** es posible asociar un menú contextual estándar al área web vía la lista de propiedades (ver [#title id="3211" anchor="429443"/]). La visualización del menú contextual puede controlarse utilizando el comando **WA SET PREFERENCE**.
- **Arrastrar y soltar:** el usuario puede arrastrar y soltar texto, imágenes y documentos dentro del área web o entre un área web y los objetos de los formularios 4D, en función de las propiedades de los objetos 4D.
  Por razones de seguridad, el cambio del contenido de un área web mediante arrastrar y soltar un archivo o una URL no está permitido por defecto a partir de 4D v14 R2. En este caso, el cursor del ratón muestra un icono de "prohibido"  . Debe usar el comando **WA SET PREFERENCE** para permitir explícitamente el soltar los URLs o los archivos en el área.

Subformularios

Por razones relacionadas con los mecanismos de redibujo de ventana, la inserción de un área Web en un subformulario está sujeta a las siguientes restricciones:

- El subformulario no debe poder desplazarse
- Los límites del área Web no deben exceder el tamaño del subformulario

Conflicto área web y servidor web (Windows)

Bajo Windows, no es recomendable acceder vía un área web al servidor web de la aplicación 4D que contiene el área porque esta configuración puede provocar un conflicto que paralice la aplicación. Por supuesto, un 4D remoto puede acceder al servidor web del 4D Server, pero no a su propio servidor web.

Plugins Web y applets Java

El uso de plugins web y applets Java **no se recomienda** en áreas web, ya que pueden conducir a inestabilidad en el funcionamiento de 4D, en particular a nivel de gestión de eventos.

Inserción del protocolo (Mac OS)

Los URLs administrados por programación en áreas web bajo Mac OS deben comenzar por el protocolo. Por ejemplo, debe pasar la cadena "http://www.misitio.com" y no únicamente "www.misitio.com".

WA Back URL available

WA Back URL available ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Booleano	↩	True si hay un URL anterior en la secuenca de URLs abiertos; de lo contrario, False

Descripción

El comando **WA Back URL available** se utiliza para saber si existe un URL precedente disponible en la secuencia de URLs abiertos en el área web designada por los parámetros * y *objeto*.

El comando devuelve **True** si existe un URL y de lo contrario **False**. Particularmente, este comando puede utilizarse en una interfaz personalizada, para activar o desactivar los botones de navegación.

WA Create URL history menu

WA Create URL history menu ({* ;} objeto {; direccion}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
direccion	Entero largo	➡	0 ó si se omite=Lista de los URLs anteriores, 1=Lista de los URLs siguientes
Resultado	MenuRef	↩	Referencia de menú

Descripción

El comando **WA Create URL history menu** crea y llena un menú que puede utilizarse directamente para la navegación entre los URLs visitados durante la sesión en el área web designada por los parámetros * y *objeto*. Puede utilizarse para crear una interfaz de navegación personalizada.

La información suminstrada concierne a la sesión; en otras palabras, la navegación se lleva a cabo en la misma área web siempre y cuando el formulario no se haya cerrado.

Pase en *direccion* un valor que indique la lista a recuperar. Puede utilizar una de las siguientes constantes, ubicadas en el tema "":

Constante	Tipo	Valor
WA next URLs	Entero largo	1
WA previous URLs	Entero largo	0

Si omite el parámetro *direccion*, se utiliza el valor 0.

Una vez generado el menú, puede mostrarlo vía el comando de 4D **Dynamic pop up menu** y puede trabajar con él utilizando los comandos estándar de gestión de menús de 4D. La cadena devuelta por el comando **Dynamic pop up menu** contiene el URL de la página visitada (ver ejemplo).

Llame el comando **RELEASE MENU** para borrar un menú de historial del URL cuando ya no sea útil.

Ejemplo

El siguiente código puede estar asociado con un botón 3D con menú pop up llamado "Anterior":

```
Case of
  `Clic simple
   :(Form event=On Clicked)
      WA OPEN BACK URL(WA_area)
  `Clic en la flecha -> mostrar pop up
   :(Form event=On Alternative Click)
  `Crear un menú de historial previo
      $Menu:=WA Create URL history menu(WA_area;WA previous URLs)
  `Mostrar este menú en un pop-up
      $URL:=Dynamic pop up menu($Menu)
  `Si un elemento está seleccionado
      If($URL#"")
  `Abrir la página Web
         WA OPEN URL(WA_area;$URL)
      End if
  `Borrar el menú para liberar la memoria
      RELEASE MENU($Menu)
End case
```

WA Evaluate JavaScript

WA Evaluate JavaScript ({* ;} objeto ; codeJS {; type}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
codeJS	Cadena	➡	Código JavaScript
type	Entero largo	➡	Tipo al cual convertir el resultado
Resultado	Fecha, Hora, Objeto, Puntero, Real, Texto	↩	Resultado de ejecución

Descripción

El comando **WA Evaluate JavaScript** ejecuta en el área web designada por los parámetros * y *objeto* el código JavaScript pasado en *codeJS* y devuelve el resultado. Este comando debe ser llamado después de cargar la página (el evento de formulario On End URL Loading debe haber sido generado).

Por defecto, el comando devuelve el resultado como cadenas. Puede utilizar el parámetro opcional *tipo* para especificar la digitación del valor devuelto. Para hacer esto, pase una de las siguientes constantes, del tema "**Tipos de campos y variables**":

Constante	Tipo	Valor
Is Boolean	Entero largo	6
Is collection	Entero largo	42
Is date	Entero largo	4
Is longint	Entero largo	9
Is object	Entero largo	38
Is real	Entero largo	1
Is text	Entero largo	2
Is time	Entero largo	11

Ejemplo 1

Este ejemplo de código JavaScript hace que se muestre el URL anterior.

```
$result:=WA Evaluate JavaScript(MyWArea;"history.back()")
```

Ejemplo 2

Este ejemplo muestra algunas evaluaciones con conversión de los valores recibidos.
Las funciones JavaScript se ubican en un archivo HTML:

```
<!DOCTYPE html> <html> <head> <script> function evalLong(){ return 123; } function
evalText(){ return "456"; } function evalObject(){ return {a:1,b:"hello world"}; }
function evalDate(){ return new Date(); } </script> </head> <body> TEST PAGE </body>
</html>
```

Escriba en el método del formulario 4D:

```
If(Form event=On Load)
   WA OPEN URL(*;"Web Area";"C:\\myDatabase\\index.html")
End if
```

Luego puede evaluar el código JavaScript desde 4D:

```
$Eval1:=WA Evaluate JavaScript(*;"Web Area";"evalLong()";Is longint)
 //$Eval1 = 123
 //$Eval1 = "123" if type is omitted
$Eval2:=WA Evaluate JavaScript(*;"Web Area";"evalText()";Is text)
 //$Eval2 = "456"
$Eval3:=WA Evaluate JavaScript(*;"Web Area";"evalObject()";Is object)
 //$Eval3 = {"a":1,"b":"hello world"}
$Eval4:=WA Evaluate JavaScript(*;"Web Area";"evalDate()";Is date)
 // $Eval4 = 06/21/13
 // $Eval4 = "2013-06-21T14:45:09.694Z" si el tipo se omite
```

WA EXECUTE JAVASCRIPT FUNCTION

WA EXECUTE JAVASCRIPT FUNCTION ({* ;} objeto ; funcionjs ; resultado|* {; param}{; param2 ; ... ; paramN})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
funcionjs	Cadena	➡	Nombre de la función JavaScript a ejecutar
resultado|*	Variable	➡	* para una función sin resultado o
		⬅	Resultado de la función (si se espera)
param	Cadena, Número, Fecha, Objeto, Collection	➡	Parámetro(s) a pasar a la función

Descripción

El comando **WA EXECUTE JAVASCRIPT FUNCTION** ejecuta en el área Web designada por los parámetros * y *objeto*, la función JavaScript *funcionJS* y devuelve opcionalmente su resultado en el parámetro *resultado*.

Si la función no devuelve un resultado, pase * en el parámetro *resultado*.

Puede pasar en *param* una o varios parámetros que contengan los parámetros de la función.
El comando soporta varios tipos de parámetros, tanto para entrada (*param*) como para salida (*resultado*). Puede pasar y recuperar datos de tipos cadena, número, fecha, objeto y colección.

Ejemplo 1

Llamada de una función JavaScript con 3 parámetros:

```
$JavaScriptFunction:="FuncionAEjecutar"
$Param1:="10"
$Param2:="true"
$Param3:="1,000.2"  `note "," como separador de miles y "." como separador décimal

WA EXECUTE JAVASCRIPT FUNCTION(MyWArea;$FuncionAEjecutar;$Result;$Param1;$Param2;$Param3)
```

Ejemplo 2

La función JavaScript "getCustomerInfo" recibe un número ID como parámetro y devuelve un objeto:

```
C_OBJECT($Result)
C_LONGINT($ID)
$ID:=1000
WA EXECUTE JAVASCRIPT FUNCTION(*;"WA";"getCustomerInfo";$Result;$ID)
```

WA Forward URL available

WA Forward URL available ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Booleano	⇐	True si existe un URL siguiente en la secuencia de URLs abiertos; de lo contrario, False

Descripción

El comando **WA Forward URL available** permite conocer si existe un URL siguiente disponible en la secuencia de URLs abiertos en el área web designada por los parámetros * y *objeto*.

El comando devuelve **True** si existe un URL y de lo contrario **False**. Particularmente, este comando puede utilizarse, en un interfaz personalizada, activar o desactivar los botones de navegación.

WA Get current URL

WA Get current URL ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (si se omite *)
Resultado	Cadena	↩	URL actualmente cargada en el área Web

Descripción

El comando **WA Get current URL** devuelve la dirección URL de la página mostrada en el área web designada por los parámetros * y *objeto*.

Si el URL actual no está disponible, el comando devuelve una cadena vacía.

Si la página web está cargada completamente, el valor devuelto por la función es idéntico al de la variable "URL" asociada con el área web. Si la página está en el proceso de ser cargada, los dos valores serán diferentes: la función devuelve el URL completamente cargado y la variable contiene el URL en proceso de ser cargado.

Ejemplo

La página mostrada es el URL "www.apple.com" y la página "www.4dhispano.com" está en proceso de ser cargada:

```
$url:=WA Get current URL(MyWArea)  `devuelve "http://www.apple.com"
  `La variable URL asociada contiene "http://www.4dhispano.com"
```

WA GET EXTERNAL LINKS FILTERS

WA GET EXTERNAL LINKS FILTERS ({* ;} objeto ; arrFiltros ; arrAutorizRechazar)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
arrFiltros	Array cadena	⬅	Array de filtros
arrAutorizRechazar	Array booleano	⬅	Array autorizar-rechazar

Descripción

El comando **WA GET EXTERNAL FILTERS LINKS** devuelve en los arrays *arrFiltros* y *arrAutorizRechazar*, los filtros de enlaces externos del área web designada por los parámetros * y *objeto*. Si ningún filtro está activo, los arrays se devuelven vacíos.

Los filtros son instalados por el comando **WA SET EXTERNAL LINKS FILTERS**. Si los arrays son reinicializados durante la sesión, el comando **WA GET EXTERNAL LINKS FILTERS** permite conocer el la configuración actual.

WA Get last filtered URL

WA Get last filtered URL ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Cadena	↩	Último URL filtrado

Descripción

El comando **WA Get last filtered URL** devuelve el último URL filtrado en el área web designada por los parámetros * y *objeto*.
El URL puede haber sido filtrado por una de las siguientes razones:

- El URL fue negado por un filtro (comando **WA SET URL FILTERS**),
- El enlace está abierto en el navegador por defecto (comando **WA SET EXTERNAL LINKS FILTERS**),
- El URL intenta abrir una ventana pop up.

Es recomendable llamar este comando en el contexto de los eventos de formulario **On URL Filtering**, **On Open External Link** y **On Window Opening Denied** con el fin de conocer el URL filtrado.

WA GET LAST URL ERROR

WA GET LAST URL ERROR ({* ;} objeto ; url ; descripcion ; codigoError)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *) o Variable (si se omite *)
url	Cadena	⇐	URL al origen del error
descripcion	Cadena	⇐	Descripción del error (Mac OS)
codigoError	Entero largo	⇐	Código de error

Descripción

El comando **WA GET LAST URL ERROR** permite recuperar varios elementos de información relacionados con el último error ocurrido en el área Web designada por los parámetros * y *objeto*.

Esta información se devuelve en tres variables:

- *url*: el URL provoca el error.
- *descripcion* (Mac OS únicamente): un texto describe el error (si está disponible). Si no es posible asociar un texto al error, se devuelve una cadena vacía. Bajo Windows, este parámetro siempre se devuelve vacío.
- *codigoError*: código del error.

- Si el código es >=400, es un error relacionado con el protocolo HTTP. Para mayor información sobre este tipo de error, consulte la siguiente dirección:

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

- De lo contrario, es un error devuelto por el WebKit (Mac OS) o ActiveX (Windows).

Es recomendable llamar este comando dentro del marco del evento de formulario **On URL Loading Error** con el fin de conocer la causa del error que acaba de ocurrir.

WA Get page content

WA Get page content ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Cadena	↩	Código HTML fuente

Descripción

El comando **WA Get page content** devuelve el código HTML de la página actual o en de la página que se va a mostrar en el área web designada por los parámetros * y *objeto*.

Este comando devuelve una cadena vacía si el contenido de la página actual no está disponible.

WA Get page title

WA Get page title ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	→	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	→	Nombre del objeto (si se especifica *) o Variable (si se omite *)
Resultado	Cadena	←	Título de la página actual

Descripción

El comando **WA Get page title** devuelve el título de la página actual o que va a ser mostrada en el área web designada por los parámetros * y *objeto*. El título corresponde a la etiqueta HTML "Title".

Este comando devuelve una cadena vacía si no hay título disponible para el URL actual.

WA GET PREFERENCE

WA GET PREFERENCE ({* ;} objeto ; selector ; valor)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
selector	Entero largo	➡	Preferencia a leer
valor	Variable	⬅	Valor actual de la preferencia

Descripción

El comando **WA GET PREFERENCE** permite obtener el valor actual de una preferencia en el área web designada por los parámetros * y *objeto*.

Pase en el parámetro *selector* la preferencia a leer. Puede pasar una de las siguientes constantes, que se encuentran en el tema **Web Area**:

Constante	Tipo	Valor	Comentario
WA enable contextual menu	Entero largo	4	Autoriza la visualización del menú contextual estándar en el área web
WA enable Java applets	Entero largo	1	Autoriza la ejecución de applets Java en el área web.
WA enable JavaScript	Entero largo	2	Autoriza la ejecución de JavaScript en el área web
WA enable plugins	Entero largo	3	Autoriza la instalación de plug-ins en el área web
WA enable URL drop	Entero largo	101	Permite soltar URLs o archivos en el área web (por defecto = false)
WA enable Web inspector	Entero largo	100	Permite la visualización del inspector web en el área

Para mayor información sobre estas preferencias, consulte la descripción del comando **WA SET PREFERENCE**.

Pase en el parámetro *valor* una variable que recibirá el valor actual de la preferencia. El tipo de la variable depende de la preferencia. La variable *valor* siempre es de tipo Booleano: contiene **True** si la preferencia está activa y si no **False**.

WA GET URL FILTERS

WA GET URL FILTERS ({* ;} objeto ; arrFiltros ; arrAutorizRechazar)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
arrFiltros	Array cadena	⬅	Array filtros
arrAutorizRechazar	Array booleano	⬅	Array Autorizar-rechazar

Descripción

El comando **WA GET URL FILTERS** devuelve en los arrays *arrFiltros* y *arrAutorizRechazar*, los filtros activos en el área web designada por los parámetros * y *objeto* . Si ningún filtro está activo, los arrays se devuelven vacíos.

Los filtros son instalados por el comando **WA SET URL FILTERS**. Si los arrays se reinicializan durante la sesión, el comando **WA GET URL FILTERS** puede utilizarse para conocer los parámetros actuales.

WA GET URL HISTORY

WA GET URL HISTORY ({* ;} objeto ; arrUrls {; direccion {; arrTitulos}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
arrUrls	Array cadena	⬅	Array de los URLs visitados
direccion	Entero largo	➡	0 ó si se omite=Lista de los URLs anteriores, 1=Lista de los URLs siguientes
arrTitulos	Array cadena	⬅	Array de títulos de ventanas

Descripción

El comando **WA GET URL HISTORY** devuelve uno o dos arrays que contienen los URLs visitados durante la sesión en el área web designada por los parámetros * y *objeto*. Permite construir una interfaz de navegación personalizada.

La información proporcionada hace referencia a la sesión; es decir la navegación efectuada en una misma área web siempre que no se haya cerrado el formulario.

El array *arrayUrls* se llena con la lista de los URLs visitados. Dependiendo del valor del parámetro *direccion* (si se pasa), el array recupera la lista de los URLs precedentes (funcionamiento por defecto) o la lista de los URLs siguientes. Esta lista corresponde al contenido de los botones estándar Anterior y Siguiente de los navegadores.

Los URLs son clasificados por orden cronológico.

Pase en *direccion* un valor indicando la lista a recuperar. Puede utilizar una de las siguientes constantes, que se encuentran en el tema "":

Constante	Tipo	Valor
WA next URLs	Entero largo	1
WA previous URLs	Entero largo	0

Si omite el parámetro *direccion*, se utiliza el valor 0.

Si se pasa, el parámetro *arrTitulos* contiene la lista de los nombres de ventanas asociados a los URLs. Este array está sincronizado con el array *arrUrls*.

WA OPEN BACK URL

WA OPEN BACK URL ({* ;} objeto)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre del objeto (si se especifica *) o Variable (si se omite *)

Descripción

El comando **WA OPEN BACK URL** carga en el área web designada por los parámetros * y *objeto* el URL precedente en la secuencia de los URLs abiertos.

Si no hay un URL precedente, el comando no hace nada. Puede probar la disponibilidad de un URL precedente con la ayuda del comando **WA Back URL available**.

WA OPEN FORWARD URL

WA OPEN FORWARD URL ({* ;} objeto)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *) o Variable (si se omite *)

Descripción

El comando **WA OPEN FORWARD URL** carga en el área web designada por los parámetros * y *objeto* el URL siguiente en la secuencia de los URLs abiertos.

Si no hay un URL siguiente (es decir, si el usuario no ha regresado al URL anterior), el comando no hace nada. Puede probar la disponibilidad de un URL siguiente utilizando el comando **WA Forward URL available**.

WA OPEN URL

WA OPEN URL ({* ;} objeto ; url)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre del objeto (si se especifica *) o Variable (si * se omite)
url	Cadena	⇒	URL a cargar en el área Web

Descripción

El comando **WA OPEN URL** carga en el área web designada por los parámetros * y *objeto* el URL pasado en el parámetro *url*.

Si se pasa una cadena vacía en *url*, el comando WA OPEN URL no hace nada y no se genera ningún error. Para cargar una página vacía en el área web, pase la cadena "about:blank" en *url*.

Como el comando **OPEN URL**, **WA OPEN URL** acepta varios tipos de sintaxis en el parámetro *url* para designar los archivos:

- sintaxis posix: "file:///c:/Mi%20Archivo"
- sintaxis sistema: "c:\MiCarpeta\MiArchivo" (Windows) o "MiDisco:MiCarpeta:MiArchivo" (Mac OS).

Nota: por compatibilidad, la sintaxis "file://" (uso de dos "/") se acepta en 4D pero no cumple con el RFC. Recomendamos utilizar la sintaxis "file:///" (con tres "/") que cumple con el RFC.

En Mac OS, cuando FileVault está activo, debe utilizar la sintaxis Posix. Puede transformar las rutas del sistema utilizando el comando **Convert path system to POSIX**.

Este comando tiene el mismo efecto que la modificación del valor de la variable "URL" asociada al área. Por ejemplo, si la variable del área se llama MiWArea_url:

```
MyWArea_url:="http://www.4d.com/"
```

Es equivalente a:

```
WA OPEN URL(MyWArea;"http://www.4d.com/")
```

WA REFRESH CURRENT URL

WA REFRESH CURRENT URL ({* ;} objeto)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *) o Variable (si se omite *)

Descripción

El comando **WA REFRESH CURRENT URL** recarga el URL actual mostrado en el área web designada por los parámetros * y *objeto*.

WA SET EXTERNAL LINKS FILTERS

WA SET EXTERNAL LINKS FILTERS ({* ;} objeto ; arrFiltros ; arrAutorizRechazar)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
arrFiltros	Array cadena	➡	Array de filtros
arrAutorizRechazar	Array booleano	➡	Array autorizar- rechazar

Descripción

El comando **WA SET EXTERNAL LINKS FILTERS** permite establecer uno o más filtros de enlaces externos para la zona web designada por los parámetros * y *objeto*. Los filtros de enlaces externos determinan si un URL asociado a la página actual vía un enlace debe abrirse en el área web o en el navegador web por defecto del equipo.

Cuando el usuario hace clic en un enlace en la página actual, 4D consulta la lista de los filtros externos con el fin de verificar el URL solicitado debe abrirse en el navegador de la máquina. Si es así, la página correspondiente al URL se muestra en el navegador web y el se genera el evento de formulario **On Open External Link**. De lo contrario (el funcionamiento por defecto), la página correspondiente al URL se muestra en el área web. La evaluación del URL está basada en el contenido de los arrays *arrFiltros* y *arrAutorizRechazar*.

Los arrays *arrFiltros* y *arrAutorizRechazar* deben estar sincronizados.

- Cada línea del array *arrFiltros* debe contener un URL a filtrar. Puede utilizar * como comodín para reemplazar uno o más caracteres.
- Cada línea correspondiente en el array *arrAutorizRechazar* debe contener un booleano indicando si el URL debe ser abierto en el área web (**True**) o en el navegador web (**False**).

En caso de contradicción a nivel de los parámetros (autorización y rechazo de un mismo URL), se tiene en cuenta la última configuración.

Para desactivar el filtro de los URL, llame el comando y pase los arrays vacío o pase, respectivamente los valores "*" y **True** en los últimos elementos de los arrays *arrFiltros* y *arrAutorizRechazar*.

Importante: el filtro establecido por el comando **WA SET URL FILTERS** se tiene en cuenta antes que el del comando **WA SET EXTERNAL LINKS FILTERS**. Esto significa que si un URL es rechazado por un filtro del comando **WA SET URL FILTERS**, no podrá abrirse en el navegador incluso si es definido explícitamente por el comando **WA SET EXTERNAL LINKS FILTERS** (ver ejemplo 2).

Ejemplo 1

Este ejemplo provoca la apertura de sitios en navegadores externos:

```
ARRAY STRING(0;$filtros;0)
ARRAY BOOLEAN($PermitirRechazar;0)

APPEND TO ARRAY($filtros;"*www.google.*")  `Seleccionar "google"
APPEND TO ARRAY($PermitirRechazar;False)
  `False: este enlace se abrirá en un navegador externo
APPEND TO ARRAY($filtros;"*www.apple.*")
APPEND TO ARRAY($PermitirRechazar;False)
  `False: este enlace se abrirá en un navegador externo
WA SET EXTERNAL LINKS FILTERS(MiWArea;$filtros;$PermitirRechazar)
```

Ejemplo 2

Este ejemplo combina los filtros de sitios y de enlaces externos:

```
ARRAY STRING(0;$filtros;0)
ARRAY BOOLEAN($PermitirRechazar;0)
APPEND TO ARRAY($filtros;"*www.google.*")  `Seleccionar "google"
APPEND TO ARRAY($PermitirRechazar;False)  `Enlace denegado
WA SET URL FILTERS(MiWArea;$filtros;$PermitirRechazar)

ARRAY STRING(0;$filtros;0)
ARRAY BOOLEAN($PermitirRechazar;0)
APPEND TO ARRAY($filtros;"*www.google.*")  `Seleccionar "google"
APPEND TO ARRAY($PermitirRechazar;False)
  `False: este enlace debe abrirse en un navegador externo pero este parámetro
  `no tiene efecto porque el enlace será bloqueado por el filtro del URL.
WA SET EXTERNAL LINKS FILTERS(MiWArea;$filtros;$PermitirRechazar)
```

WA SET PAGE CONTENT

WA SET PAGE CONTENT ({* ;} objeto ; contenido ; baseURL)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *) o Variable (si se omite *)
contenido	Cadena	⇒	Código HTML fuente
baseURL	Cadena	⇒	URL para las referencias relativas (Mac OS)

Descripción

El comando **WA SET PAGE CONTENT** reemplaza la página mostrada en el área web designada por los parámetros * y *objeto* por el código HTML pasado en el parámetro *contenido*.

El parámetro *baseURL* permite definir bajo Mac OS, un URL de base que se añadirá en frente de los enlaces relativos que se puedan encontrar en la página.

Bajo Windows, este parámetro no tiene efecto y el URL de base no está definido. Por lo tanto no es posible utilizar referencias relativas en esta plataforma.

Nota: bajo Windows, es imperativo que una página haya sido cargada en el área web antes de que se llame este comando. Si es necesario, puede pasar el URL "about:blank" con el fin de cargar una página vacía.

Ejemplo

Mostrar la frase "¡Hola mundo!" y definición de un URL de base "file:///" base URL (Mac OS únicamente):

```
WA SET PAGE CONTENT(MiWArea;"<html><body><h1>Hola Mundo!</h1></body></html>";"file:///")
```

WA SET PAGE TEXT LARGER

WA SET PAGE TEXT LARGER ({* ;} objeto)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *) o Variable (si se omite *)

Descripción

El comando **WA SET PAGE TEXT LARGER** aumenta el tamaño del texto mostrado en el área web designada por los parámetros * y *objeto*.

Bajo Mac OS, el alcance de este comando es la sesión 4D: la configuración efectuada por este comando no se conserva después del cierre de la aplicación 4D.

Bajo Windows, el alcance de este comando es global: la configuración se conserva después del cierre de la aplicación 4D.

WA SET PAGE TEXT SMALLER

WA SET PAGE TEXT SMALLER ({* ;} objeto)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)

Descripción

El comando **WA SET PAGE TEXT SMALLER** reduce el tamaño del texto mostrado en el área web designada por los parámetros * y objeto.

Bajo OS, el alcance de este comando es la sesión 4D: la configuración efectuada por este comando no se conserva después de cerrar la aplicación 4D.

Bajo Windows, el alcance de este comando es global: la configuración se conserva después de cerrar la aplicación 4D.

WA SET PREFERENCE

WA SET PREFERENCE ({* ;} objeto ; selector ; valor)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *) o Variable (si se omite *)
selector	Entero largo	⇒	Preferencia a modificar
valor	Booleano	⇒	Valor de la preferencia (True = permitido, False = no autorizado)

Descripción

El comando **WA SET PREFERENCE** permite fijar diferentes preferencias para el área web designada por los parámetros * y *objeto*.

Pase en el parámetro *selector* la preferencia a modificar y en *valor* el valor a atribuirle. Puede pasar en *selector*, una de las siguientes constantes, que se encuentran en el tema **Web Area**:

Constante	Tipo	Valor	Comentario
WA enable contextual menu	Entero largo	4	Autoriza la visualización del menú contextual estándar en el área web
WA enable Java applets	Entero largo	1	Autoriza la ejecución de applets Java en el área web.
WA enable JavaScript	Entero largo	2	Autoriza la ejecución de JavaScript en el área web
WA enable plugins	Entero largo	3	Autoriza la instalación de plug-ins en el área web
WA enable URL drop	Entero largo	101	Permite soltar URLs o archivos en el área web (por defecto = false)
WA enable Web inspector	Entero largo	100	Permite la visualización del inspector web en el área

Para cada preferencia, pase **True** en *valor* para activarla y **False** para desactivarla.

Nota: el uso de plugins web y applets Java **no se recomienda** en áreas web, ya que puede conducir a inestabilidad en el funcionamiento de 4D, en particular a nivel de la gestión de eventos.

Ejemplo

Para activar la URL suéltela en el área web 'myarea':

```
WA SET PREFERENCE(*;"myarea";WA enable URL drop;True)
```

WA SET URL FILTERS

WA SET URL FILTERS ({* ;} objeto ; arrFiltros ; arrAutorizRechazar)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
arrFiltros	Array cadena	➡	Array de filtros
arrAutorizRechazar	Array booleano	➡	Array autorizar- rechazar

Descripción

El comando WA SET URL FILTERS permite colocar uno o más filtros para el área Web designada por los parámetros * y *objeto*.

Antes de cargar toda página solicitada, 4D consulta la lista de filtros con el fin de verificar si el URL objetivo está permitido. La evaluación del URL está basada en los contenidos de los arrays *arrFiltros* y *arrAutorizRechazar*.

Si el URL solicitado no está autorizado, no se carga y se genera el evento de formulario On URL Filtering.

Los arrays *arrFiltros* y *arrAutorizRechazar* deben estar sincronizados.
• Cada elemento del array *arrFiltros* debe contener un URL a filtrar. Puede utilizar * como comodín para reemplazar uno o más caracteres.
• Cada elemento correspondiente en el array *arrAutorizRechazar* debe contener un booleano indicando si el URL debe ser autorizado (**True**) o rechazado (**False**).

En caso de contradicción a nivel de los parámetros (autorización y rechazo de un mismo URL), se tendrá en cuenta la última configuración.

Para desactivar el filtro de los URLs, llame el comando y pase arrays vacíos o pase, respectivamente, los valores "*" y **True** en los últimos elementos de los arrays *arrFiltros* y *arrAutorizRechazar*.

Una vez ejecutado el comando, los filtros se vuelven propiedad del área Web. Si los arrays *arrFiltros* y *arrAutorizRechazar* son borrados o reinicializados, los filtros permanecen activos siempre que el comando no haya sido ejecutado nuevamente. Para conocer los filtros activos para un área, debe utilizar el comando **WA GET URL FILTERS**.

Importante: el filtro de los URLs efectuado por este comando sólo aplica a la variable "URL" primaria de la página, bien sea del usuario, código javascript o código 4D, excepto para el comando **WA OPEN URL** y las URLs que comienzan con "javascript:".

Ejemplo 1

Usted quiere negar el acceso a todos los sitios web .org, .net y .fr:

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*.org")
APPEND TO ARRAY($AllowDeny;False)
APPEND TO ARRAY($filters;"*.net")
APPEND TO ARRAY($AllowDeny;False)
APPEND TO ARRAY($filters;"*.fr")
APPEND TO ARRAY($AllowDeny;False)
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

Ejemplo 2

Usted quiere negar el acceso a todos los sitios Web excepto los sitios rusos (.ru):

```
ARRAY TEXT($filtros;0)
ARRAY BOOLEAN($PermitirRechazar;0)

APPEND TO ARRAY($filtros;"*")  `Seleccionar todo
APPEND TO ARRAY($PermitirRechazar;False)  `Negar todo
APPEND TO ARRAY($filtros;"www.*.ru")  `Seleccionar *.ru
APPEND TO ARRAY($PermitirRechazar;True)  `Permitir

WA SET URL FILTERS(MiWArea;$filtros;$PermitirRechazar)
```

Ejemplo 3

Usted quiere permitir el acceso únicamente a los sitios Web 4D (.com, .fr, .es, etc.):

```
ARRAY TEXT($filtros;0)
ARRAY BOOLEAN($PermitirRechazar;0)
```

```
APPEND TO ARRAY($filtros;"*")  `Seleccionar todo
APPEND TO ARRAY($PermitirRechazar;False)  `Rechazar todo
APPEND TO ARRAY($filtros;"www.4D.*")  `Seleccionar 4d.fr, 4d.com...
APPEND TO ARRAY($PermitirRechazar;True)  `Permitir
WA SET URL FILTERS(MiWArea;$filtros;$PermitirRechazar)
```

Ejemplo 4

Usted quiere autorizar el acceso local a la documentación únicamente (ubicada en la carpeta C://doc):

```
ARRAY TEXT($filtros;0)
ARRAY BOOLEAN($PermitirRechazar;0)
```

```
APPEND TO ARRAY($filtros;"*")  `Seleccionar todo
APPEND TO ARRAY($PermitirRechazar;False)  `Negar todo
APPEND TO ARRAY($filtros;"file://C:/doc/*")
  `Seleccionar la ruta al archivo:// autorizado
APPEND TO ARRAY($PermitirRechazar;True)  `Autorizar
```

```
WA SET URL FILTERS(MiWArea;$filtros;$PermitirRechazar)
```

Ejemplo 5

Usted quiere autorizar todos los sitios excepto uno, por ejemplo el sitio Elcaro:

```
ARRAY TEXT($filtros;0)
ARRAY BOOLEAN($PermitirRechazar;0)
```

```
APPEND TO ARRAY($filtros;"*")
APPEND TO ARRAY($PermitirRechazar;True)  `Permitir todos
APPEND TO ARRAY($filtros;"*elcaro*")  `Negar todo el contenido del elcaro
APPEND TO ARRAY($PermitirRechazar;False)
WA SET URL FILTERS(MiWArea;$filtros;$PermitirRechazar)
```

Ejemplo 6

Usted quiere negar el acceso a direcciones IP específicas:

```
ARRAY TEXT($filtros;0)
ARRAY BOOLEAN($PermitirRechazar;0)
```

```
APPEND TO ARRAY($filtros;"*")  `Seleccionar todo
APPEND TO ARRAY($PermitirRechazar;True)  `Permitir todo
APPEND TO ARRAY($filtros;86.83.*")  `Seleccionar las IP que comienzan por 86.83.
APPEND TO ARRAY($PermitirRechazar;False)  `Negar
APPEND TO ARRAY($filtros;86.1*")  `Seleccionar las IP que comienzan por 86.1 (86.10, 86.135 etc.)
APPEND TO ARRAY($PermitirRechazar;False)  `Negar
```

```
WA SET URL FILTERS(MiWArea;$filtros;$PermitirRechazar)
  `(Note que la dirección IP de un dominio puede variar).
```

WA STOP LOADING URL

WA STOP LOADING URL ({* ;} objeto)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *) o Variable (si se omite *)

Descripción

El comando **WA STOP LOADING URL** detiene la carga de los recursos del URL actual de la zona web designada por los parámetros * y *objeto*.

Arrastrar y soltar

- Arrastrar y soltar
- Método de base On Drop
- DRAG AND DROP PROPERTIES
- Drop position
- SET DRAG ICON

Arrastrar y soltar

4D dispone de funciones integradas que le permiten arrastrar y soltar objetos en sus formularios y aplicaciones. Es posible arrastrar y soltar un objeto sobre otro, ubicado en la misma ventana o en otra ventana. En otras palabras, usted puede arrastrar y soltar al interior de un proceso o entre diferentes procesos.

Igualmente usted puede arrastrar y soltar objetos entre formularios 4D y otras aplicaciones y viceversa. Por ejemplo, es posible arrastrar y soltar un archivo de imagen GIF en un campo imagen 4D. También es posible seleccionar texto en una aplicación de procesamiento de palabras y soltarla en una variable de texto 4D.

Finalmente, es posible soltar objetos directamente en la aplicación sin tener necesariamente un formulario de fondo. El **Método de base On Drop** puede utilizarse para administrar la acción de arrastrar y soltar en este caso. Esto significa, por ejemplo, que puede abrir un documento 4D Write soltándolo en el icono de la aplicación 4D

Nota: como introducción, asumimos que una acción arrastrar y soltar "transporta" datos de un punto a otro. Más adelante, veremos que arrastrar y soltar también puede ser una metáfora ara todo tipo de operación.

Propiedades de objetos "Arrastrable" y "Soltable"

Para arrastrar y soltar un objeto sobre otro objeto, debe seleccionar la **Propiedad arrastrable** para ese objeto en la ventana de la Lista de propiedades. En una operación arrastrar y soltar, el objeto que se arrastra es el **objeto fuente**.

Para hacer que un objeto sea soltable, es decir que el objeto pueda ser el destino de una operación de arrastrar y soltar, debe seleccionar la **propiedad soltable** para ese objeto en la Lista de propiedades. En una operación arrastrar y soltar, el objeto que recibe los datos es el **objeto de destino**.

Arrastrar automático y **Soltar automático:**
Estas propiedades adicionales están disponibles para campos y variables de tipo texto, combo boxes y list boxes. La opción **Soltar automático** también está disponible para campos y variables tipo imagen. Pueden utilizarse para activar un modo arrastrar y soltar automático basado en copiar el contenido (la acción arrastrar y soltar ya nos es administrada por los eventos de formulario 4D). Por favor consulte el párrafo "Arrastrar y saltar automático" al final de esta sección.

Por defecto, los objetos creados recientemente no poseen ninguna de estas propiedades. Es su decisión seleccionar estas propiedades.

Todos los objetos en un formulario de entrada o en un diálogo, pueden ser definidos como arrastrables y soltables. Los elementos individuales de un array (por ejemplo, un área de desplazamiento), los elementos de una lista jerárquica o las filas en una list box pueden ser arrastrables y soltables. Por el contrario, usted puede arrastrar y soltar un objeto sobre un elemento individual de un array o de una lista jerárquica o una línea de un list box. Sin embargo, no puede arrastrar y soltar objetos del área de detalle de un formulario de salida.

Igualmente puede administrar arrastrar y soltar en la aplicación, fuera de todos los formularios, utilizando el **Método de base On Drop.**

Fácilmente puede crear una interfaz de usuario arrastrar y soltar, porque 4D le permite utilizar todo tipo de objeto activo (campo o variable) como objetos fuente o de destino. Por ejemplo, usted puede arrastrar y soltar un botón.

Notas:

- Para arrastrar un texto o un botón que tiene la propiedad "arrastrable," primero debe presionar la tecla **Alt** (Windows) u **Opción** (Mac OS).
- Por defecto, en el caso de campos y variables de imagen, la imagen y su referencia son arrastrados. Si usted sólo quiere arrastrar la referencia de la variable o campo, primero presione la tecla **Alt** (Windows) u **Opción** (Mac OS).
- Cuando en un objeto de tipo List box, las propiedades "Arrastrable" y "Filas desplazables" están definidas simultáneamente, la propiedad "Filas desplazables" tiene prioridad cuando se mueve una fila. En este caso no es posible arrastrarlo.

Un objeto que es arrastrable y soltable también puede ser soltable en sí mismo, a menos de que rechace la operación. Para más detalles, consulte los párrafos siguientes.

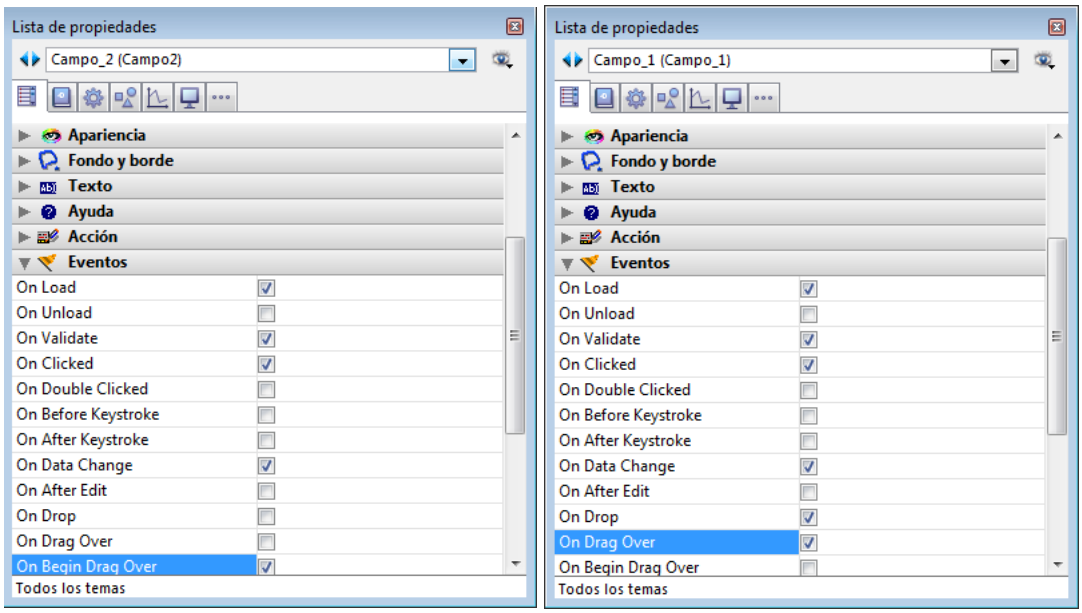
La siguiente imagen muestra la ventana de la Lista de propiedades con las propiedades Arrastrable y Soltable definidas para el objeto seleccionado:

Administrar por programación arrastrar y soltar

La gestión de arrastrar y soltar por programación está basada en tres eventos de formulario: **On Begin Drag Over, On Drag Over** y **On Drop**.

Note que el evento **On Begin Drag Over** se genera en el contexto del objeto fuente del arrastrar mientras **On Drag Over** y **On Drop** se envían al objeto de destino únicamente.

Para que la aplicación procese estos eventos, deben haber sido seleccionados de una forma adecuada en la Lista de Propiedades:



On Begin Drag Over

El evento de formulario On Begin Drag Over es seleccionable para todos los objetos de formulario que puedan arrastrarse. Se genera en todos los casos donde el objeto tenga la propiedad **Arrastrable**.

A diferencia del evento de formulario On Drag Over, On Begin Drag Over se llama dentro del contexto del objeto fuente de la acción arrastrar. Puede llamarse desde el método del objeto fuente o desde el método del formulario del objeto fuente.

Este evento es útil para la gestión avanzada de la acción arrastrar. Puede utilizarse para:

- Obtener los datos y las firmas presentes en el portapapeles (por el comando **GET PASTEBOARD DATA**).
- Añadir los datos y las firmas al portapapeles (por el comando **APPEND DATA TO PASTEBOARD**)
- Utilizar un ícono personalizado durante la acción de arrastrar (vía el comando **SET DRAG ICON**).
- Aceptar o rechazar el arrastrar vía *$0* en el método del objeto arrastrado. Para indicar que la acción arrastrar es aceptada, el método del objeto fuente debe devolver 0 (cero); por lo tanto usted debe ejecutar *$0:=0*. Para indicar que las acciones de arrastre son rechazadas, el método o el objeto fuente debe devolver -1 (menos 1); usted debe por lo tanto ejecutar *$0:=-1*. Si no se devuelve resultado, 4D considera que el arrastrar es aceptado.

Los datos de 4D se colocan en el portapapeles antes de llamar al evento. Por ejemplo, en el caso de arrastre sin la acción **Arrastre automático**, el texto arrastrado ya está en el portapapeles cuando se llama el evento.

On Drag Over

El evento **On Drag Over** se envía repetidamente al objeto de destino cuando el puntero del ratón se mueve sobre el objeto. En respuesta a este evento, usted generalmente:

- Llama al comando **DRAG AND DROP PROPERTIES**, el cual le informa sobre el objeto fuente.
- Dependiendo de la naturaleza y tipo del objeto de destino (cuyo método de objeto está siendo ejecutado) y el objeto fuente, usted **acepta** o **rechaza** el arrastrar y soltar.

Para aceptar arrastrar, el método del objeto de destino debe devolver *0* (cero), de manera que usted escribe *$0:=0*. Para rechazar el arrastrar, el método de objeto debe devolver *-1* (menos uno), de manera que usted escribe *$0:=-1*. Durante un evento **On Drag Over,** 4D trata el método del objeto como una función. Si no devuelve ningún resultado, 4D asume que se acepta arrastrar.

Si acepta el arrastrar, el objeto de destino se activa. Si lo rechaza, el objeto de destino permanece inactivo. Aceptar el arrastrar no significa que los datos arrastrados sean insertados en el objeto de destino. Sólo significa que si el botón del ratón fue liberado en este punto, el objeto de destino aceptaría los datos arrastrados.

Si no procesa el evento **On Drag Over** para un objeto soltable, el objeto se activará para todas las operaciones de arrastrar, sin importar la naturaleza y tipo de los datos arrastrados.

El evento **On Drag Over** le permite controlar la primera fase de una operación de arrastrar y soltar. No sólo puede probar si el tipo de los datos arrastrados es compatible con el objeto de destino y luego aceptar o rechazar el arrastre; usted puede simultáneamente notificar al usuario de este hecho, porque 4D activa o no el objeto de destino, en función de su decisión.

El código que trata un evento **On Drag Over** debe ser corto y ejecutarse rápidamente, porque el evento es enviado repetidamente al objeto actual de destino, debido a los movimientos del ratón.

Advertencia: a partir de la versión 11 de 4D, si el arrastrar y soltar es un **arrastrar y soltar interproceso**, lo cual significa que el objeto fuente está ubicado en un proceso (ventana) diferente de la del objeto de destino, el método del objeto de destino para un evento **On Drag Over** se ejecuta **en el contexto del proceso de destino.** Para conocer el valor de los elementos arrastrados, debe utilizar los comandos de comunicación interproceso. Generalmente es preferible en ese caso utilizar el evento On Begin Drag Over y los comandos del tema .

On Drop

El evento **On Drop** se envía al objeto de destino (una sola vez) cuando el puntero del ratón se libera sobre el objeto. Este evento es la segunda fase de la operación arrastrar y soltar, en la cual usted realiza una operación en respuesta a la acción del usuario.

Este evento no se envía al objeto si el arrastrar no se ha aceptado durante los eventos **On Drag Over**. Si procesa el evento **On Drag Over** para un objeto y rechaza un arrastrar, no ocurre el evento **On Drop**. Si durante el evento **On Drag Over** usted ha probado la compatibilidad de tipos de datos entre los objetos fuente y destino y ha aceptado un posible soltar, no necesita probar nuevamente los datos durante **On Drop**. Usted ya sabe que los datos son compatibles con el objeto de destino.

Un aspecto interesante de la implementación de arrastrar y soltar es que 4D le permite hacer lo que usted quiera. Ejemplos:

- Si un elemento de una lista jerárquica se suelta sobre un campo de tipo texto, usted puede insertar el texto del elemento de la lista al comienzo, al final, o en la mitad del campo de texto.
- Su formulario contiene un botón de imagen de dos estados, el cual representa una canasta vacía o llena. Soltar un objeto sobre ese botón podría significar (desde el punto de vista de la interfaz del usuario) "suprimir el objeto que ha sido arrastrado y soltado en la canasta." Acá, arrastrar y soltar no transporta datos de un punto a otro; en lugar de eso, realiza una acción.
- Arrastrar un elemento de un array de una ventana flotante a un objeto en un formulario podría significar "mostrar en esta ventana el registro del cliente cuyo nombre usted acaba de arrastrar y soltar desde la ventana flotante listando los clientes almacenados en la base de datos".
- Etc.

La interfaz de arrastrar y soltar de 4D es un framework que le permite a cualquier usuario implementar todas las metáforas de interfaz de usuario que usted pueda imaginar.

Comandos del tema arrastrar y soltar

Los comandos **DRAG AND DROP PROPERTIES** devuelven:

- Un puntero hacia el objeto arrastrado (campo o variable)
- El número de elemento, si el objeto arrastrado es un elemento del Array o un elemento de una lista
- El número del proceso fuente.

El comando **Drop position** devuelve el número o la posición del elemento objetivo si el objeto de destino es un array (es decir, un área de desplazamiento) una lista jerárquica, un texto o un combo box, como también el número de columna si el objeto es una list box.

Los comandos como **RESOLVE POINTER** y **Type** son útiles para probar la naturaleza y el tipo de objeto fuente.

Cuando la operación arrastrar y soltar está destinada a copiar los datos arrastrados, la funcionalidad de estos comandos depende de cuántos procesos estén involucrados:

- Si el arrastrar y soltar está limitado a un proceso, utilice estos comandos para realizar las acciones correspondientes (es decir, simplemente asignar el objeto fuente al objeto de destino).
- Si el arrastrar y soltar es interproceso, debe tener cuidado cuando acceda a los datos arrastrados; debe acceder a la instancia de datos del proceso fuente. Si los datos arrastrados provienen de una variable, utilice **GET PROCESS VARIABLE** para obtener el valor correcto. Si los datos arrastrados provienen de un campo, recuerde que el registro actual de una tabla probablemente no es el mismo para los dos procesos, de manera que debe acceder al registro correcto. Por lo general se recomienda en este caso utilizar los comandos del tema y el evento On Begin Drag Over.

Si el arrastrar y soltar no está destinado a mover datos, sino a crear metáforas de interfaz para una operación en particular, usted puede hacer todo lo que quiera.

Comandos del tema Portapapeles

Si las operaciones arrastrar y soltar implican el movimiento de datos heterogéneos o de documentos entre dos aplicaciones 4D o una aplicación 4D y una tercera aplicación, los comandos del tema "Portapapeles" le ofrecerán las herramientas necesarias.

De hecho, estos comandos puede utilizarse para administrar el copiar/pegar y arrastrar y soltar de datos. 4D utiliza dos portapapeles: uno para datos copiados (o cortados), el cual es el portapapeles actual, y el otro para los datos que están siendo arrastrados y soltados. Estos dos portapapeles son administrados utilizando los mismos comandos. Usted accede a uno o a otro dependiendo del contexto.

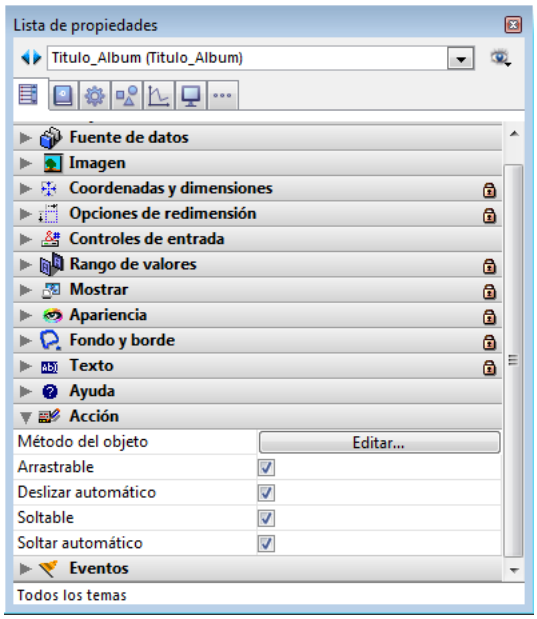
Para mayor información sobre la utilización de los comandos del tema "Portapapeles" para operaciones de arrastrar y soltar, por favor consulte la sección .

Arrastar y soltar automático

Las áreas de texto (campos, variables, combo boxes y list boxes) como también objetos de imagen permiten el arrastrar y soltar automático, es decir el movimiento o copia de una selección de texto o imagen de un área a otra en un solo clic. Puede utilizarse en la misma área 4D, entre dos áreas 4D, o entre 4D y otra aplicación, por ejemplo WordPad.

Nota: en el caso de arrastrar y soltar automático entre dos áreas 4D, los datos son desplazados, en otras palabras, son eliminados del área fuente. Si quiere volver a copiar los datos, mantenga presionadas **Ctrl** (Windows) u **Opción** (OS X) durante la acción (bajo OS X, debe presionar la tecla **Opción** luego de iniciar arrastrar los elementos).

El arrastrar y soltar automático puede ser configurado por separado para cada objeto de un formulario vía dos opciones de la Lista de propiedades: **Arrastrar automático** y **Soltar automático**.



- **Arrastrar automático** (objetos de tipo texto únicamente):cuando esta opción está seleccionada, el modo arrastrar automático se activa para el objeto. Este modo es prioritario para las imágenes, incluso si la opción **Arrastrable** está seleccionada. En este modo, el evento de formulario On Begin Drag Over NO se genera. Si desea "forzar" la utilización del arrastrar estándar cuando el arrastrar automático esté activo, presione la tecla **Alt** (Windows) u **Opción** (OS X) durante la operación (bajo OS X, debe presionar la tecla **Opción** luego de iniciar arrastrar los elementos).
  Esta opción no está disponible para las imágenes.

- **Soltar automático**: Esta opción se utiliza para activar el modo soltar automático. En este modo, 4D maneja automáticamente, si es posible, la inserción de datos arrastrados de tipo texto o imagen y soltados en el objeto (los datos son pegados en el objeto). Los eventos de formulario On Drag Over y On Drop no son generados en este caso.
  En el caso de soltar datos diferentes de texto o imágenes (otro objeto 4D, archivo, etc.) o datos complejos, la aplicación se refiere al valor de la opción **Soltable**: si está seleccionada, se generan los eventos de formulario On Drag Over y On Drop; de lo contrario, se rechaza el soltar. Esto también depende del valor de la opción "Prohibido soltar datos que no provengan de 4D" (ver a continuación).

Prohibido soltar datos que no provengan de 4D (compatibilidad)

A partir de la versión 11, 4D permite el arrastrar y soltar de selecciones, de objetos, o de archivos externos a 4D, como por ejemplo los archivos de imágenes. Esta posibilidad debe estar soportada por el código de la base.

En las bases convertidas de una versión anterior de 4D, esta posibilidad puede llevar a malfuncionamiento si el código existente no está adaptado. Por esta razón, una opción en las Preferencias puede utilizarse para desactivar esta función: **Prohibido soltar datos que no provengan de 4D.**
Esta opción se encuentra en la página Aplicación/Compatibilidad. Está seleccionada por defecto en las bases convertidas.

Cuando esta opción está seleccionada, se rechaza el soltar objetos externos en formularios 4D. Note sin embargo que la inserción de objetos externos continua siendo posible en objetos que tengan la opción **Soltar automático**, cuando la aplicación puede interpretar los datos soltados (texto o imagen).

Método de base On Drop

El **Método de base On Drop** está disponible en aplicaciones 4D locales o remotas.

Este método de base se ejecuta automáticamente en el caso de soltar objetos en la aplicación 4D fuera de un contexto formulario o dialogo. Dependiendo de la plataforma y del tipo de aplicación, se soportan diferentes tipos de soltar:

Acción	Plataforma(s)	Comentario
Soltar en un área vacía de la ventana MDI	Windows	No disponible cuando la base se ejecuta en modo SDI ya que no hay ninguna ventana MDI en este contexto (consulte la sección **Modo SDI en Windows**).
Soltar en el icono 4D en el Dock	macOS	
Soltar en el icono 4D en el escritorio del sistema	Windows(*) y macOS	**Método de base On Drop** se llama únicamente, si la aplicación ya ha sido lanzada, excepto en el caso de las aplicaciones fusionadas con 4D Desktop. En este caso, el método base se llama incluso cuando la base no ha sido lanzada. Esto significa que es posible definir firmas de documentos personalizadas.

(*) No compatible con 4D Developer 64 bits en Windows porque esta acción inicia una nueva instancia de aplicación (funcionalidad del sistema).

Bajo Mac OS, debe presionar las teclas **Opción**+**Comando** al soltar para que se llame el método de base.

Ejemplo

Este ejemplo puede ser utilizado para abrir un documento 4D Write que fue soltado fuera de un formulario:

```
  `Método base On Drop
archivoSoltado:=Get file from pasteboard(1)
If(Position(".4W7";droppedFile)=Length(archivoSoltado)-3)
   areaExterna:=Open external window(100;100;500;500;0;archivoSoltado;"_4D Write")
   WR OPEN DOCUMENT(areaExterna;archivoSoltado)
End if
```

DRAG AND DROP PROPERTIES

DRAG AND DROP PROPERTIES (srcObjeto ; srcElemento ; srcProceso)

Parámetro	Tipo		Descripción
srcObjeto	Puntero	⇐	Puntero hacia el objeto fuente de arrastrar y soltar
srcElemento	Entero largo	⇐	Número del elemento del array arrastrado o Número de la fila de la list box arrastrada o Elemento de la lista jerárquica arrastrada o -1 si el objeto arrastrado no es ni un elemento de array ni de list box ni de lista jerárquica
srcProceso	Entero largo	⇐	Número de proceso fuente

Descripción

El comando **DRAG AND DROP PROPERTIES** permite obtener información sobre el objeto fuente cuando un evento **On Drag Over** u **On Drop** ocurre para un objeto "complejo" (array, list box o lista jerárquica).

Generalmente, se utiliza **DRAG AND DROP PROPERTIES** desde dentro del método de objeto del objeto (o desde una de las subrutinas que llama) para el cual el evento **On Drag Over** u **On Drop** occure (el objeto de destino).

Importante: los datos pueden soltarse en un objeto de formulario si la propiedad **Soltable** ha sido seleccionada. Igualmente, su método de objeto debe ser activado por **On Drag Over** y/u **On Drop**, para procesar estos eventos.

Después de la llamada:

- El parámetro *srcObjeto* es un puntero hacia el objeto fuente (el objeto que ha sido arrastrado y soltado). Note que este objeto puede ser el objeto de destino (el objeto para el cual el evento **On Drag Over** u **On Drop** occure) o un objeto diferente. Arrastrar y soltar datos desde y hacia el mismo objeto es útil en los arrays y listas jerárquicas, es una manera simple de permitir al usuario ordenar un array o un lista manualmente.
- Si los datos arrastrados y soltados son un elemento de un array (siendo el objeto fuente un array), el parámetro *srcElemento* devuelve el número de este elemento. Si los datos arrastrados y soltados son una fila de un list box, el parámetro *srcElemento* devuelve el número de esta fila. Si los datos arrastrados y soltados son un elemento de lista (siendo el objeto fuente una lista jerárquica), el parámetro *srcElemento* devuelve la posición de este elemento. De lo contrario, si el objeto fuente no pertenece a ninguna de estas categorías, *srcElemento* es igual a -1.
- Las operaciones arrastrar y soltar pueden ocurrir entre procesos. El parámetro *srcProceso* es igual al número del proceso al cual pertenece el objeto fuente. Es importante probar el valor de este parámetro. Puede responder a un arrastrar y soltar dentro del mismo proceso simplemente copiando los datos fuente en el objeto de destino. Por otra parte, cuando está tratando un arrastrar y soltar interproceso, debe utilizar el comando **GET PROCESS VARIABLE** para obtener los datos fuente a partir de la instancia del objeto del proceso fuente. Generalmente, el arrastrar y soltar en una interfaz usuario se efectúa desde las variables (arrays y listas) hacia las áreas de entrada de datos (campos o variables).

Nota de compatibilidad: a partir de la versión 11 de 4D, se recomienda administrar las operaciones de arrastrar y soltar, especialmente interproceso, con la ayuda del evento On Begin Drag Over y de los comandos del tema .

Si llama a **DRAG AND DROP PROPERTIES** cuando no hay ningún evento arrastrar y soltar, *srcObjeto* devuelve un puntero NIL, *srcElemento* devuelve -1 y *srcProceso* devuelve 0.

Consejo: 4D administra automáticamente el aspecto gráfico de arrastrar y soltar. Entonces usted debe responder al evento de manera apropiada. En los siguientes ejemplos, la respuesta es copiar los datos que han sido arrastrados. De manera alternativa, puede implementar interfaces de usuario sofisticadas donde, por ejemplo, arrastrar y soltar un elemento de array de una ventana flotante hace que la ventana de destino se llene (la ventana donde el objeto de destino está ubicado) con datos estructurados (como varios campos que provienen de un registro único identificado por el elemento de array fuente).

Se utiliza **DRAG AND DROP PROPERTIES** durante un evento **On Drag Over** para decidir si el objeto de destino acepta la operación arrastrar y soltar, dependiendo del tipo y/o la naturaleza del objeto fuente (o de cualquier otra razón). Si acepta arrastrar y soltar, el método de objeto debe devolver *$0:=0*. Si no acepta arrastrar y soltar, el método de objeto debe devolver *$0:=-1*. La aceptación o rechazo de arrastrar y soltar se refleja en la pantalla, el objeto se resalta o no como destino potencial de la operación arrastrar soltar.

Ejemplo 1

En varios de los formularios de su base, hay áreas de desplazamiento donde usted quiere reordenar manualmente los elementos simplemente arrastrándolos y soltándolos al interior de cada área. En lugar de escribir código específico para cada caso, puede implementar un método de proyecto genérico que maneje todas las áreas de desplazamiento. Puede escribir un código como este:

```
` Método de proyecto Manejo arrastrar y soltar interno en un array
` Manejo arrastrar y soltar interno en un array ( Puntero ) -> Booleano
` Manejo arrastrar y soltar interno en un array ( -> Array ) -> Es un arrastrar y soltar interno en un array
Case of
   :(Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
      If($vpSrcObj=$1)
` Aceptar arrastrar y soltar si es interno del array
         $0:=0
      Else
         $0:=-1
      End if
   :(Form event=On Drop)
```

```
   ` Obtener la información sobre el objeto fuente de arrastrar y soltar
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
   ` Obtener el número del elemento de destino
      $vlDstElem:=Drop position
   ` Si el elemento no fue soltado sobre si mismo
      If($vlDstElem #$vlSrcElem)
   ` Guardar el elemento arrastrado en el elemento 0 del array
         $1->{0}:=$1->{$vlSrcElem}
   ` Borrar el elemento arrastrado
         DELETE FROM ARRAY($1->;$vlSrcElem)
   ` Si el elemento de destino estaba más allá del elemento arrastrado
         If($vlDstElem>$vlSrcElem)
   ` Decremento del número del elemento de destino
            $vlDstElem:=$vlDstElem-1
         End if
   ` Si arrastrar y soltar ocurre más allá del último elemento
         If($vlDstElem=-1)

   ` Definir el número del elemento de destino para un nuevo elemento al final del array $vlDstElem:=Size of array($1->)+1
         End if
   ` Insertar este nuevo elemento
         INSERT IN ARRAY($1->;$vlDstElem)
   ` Fijar su valor el cual fue almacenado previamente en el elemento cero del array
         $1->{$vlDstElem}:=$1->{0}
   ` El elemento se convierte en el nuevo elemento seleccionado del array
         $1->:=$vlDstElem
      End if End case
```

Una vez haya implementado este método de proyecto, puede utilizarlo de la siguiente forma:

```
   ` Método de objeto del área de desplazamiento anArray

Case of
   `...
   :(Form event=On Drag Over)
      $0:=Manejo arrastrar y soltar interno en un array(Self)
   :(Form event=On Drop)
      Manejo arrastrar y soltar interno en un array(Self)
   `...
End case
```

Ejemplo 2

En varios de los formularios de su base, tiene áreas de texto editables en las cuales quiere arrastrar y soltar datos de varias fuentes. En lugar de escribir código específico para cada caso, puede implementar un método de proyecto genérico que maneje todas las áreas de texto editables. Puede escribir el método siguiente:

```
   ` Método de proyecto Tratamiento de soltar en variable Texto
   ` Tratamiento de soltar en variable Texto ( Puntero )
   ` Tratamiento de soltar en variable Texto ( -> variable texto o cadena )

Case of
   ` Uso de este evento para aceptar o rechazar el arrastrar y soltar
   :(Form event=On Drag Over)
   ` Inicializar $0 para rechazar
      $0:=-1
   ` Obtener la información sobre el objeto fuente de arrastrar y soltar
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
   ` En este ejemplo, no permitimos arrastrar y soltar desde un objeto hacia sí mismo
      If($vpSrcObj # $1)
   ` Obtener el tipo de los datos arrastrados
         $vlSrcType:=Type($vpSrcObj->)
         Case of
            :($vlSrcType=ls text)
   ` OK para las variables texto
               $0:=0
            :($vlSrcType=ls string var)
   ` OK para las variables cadena
               $0:=0
```

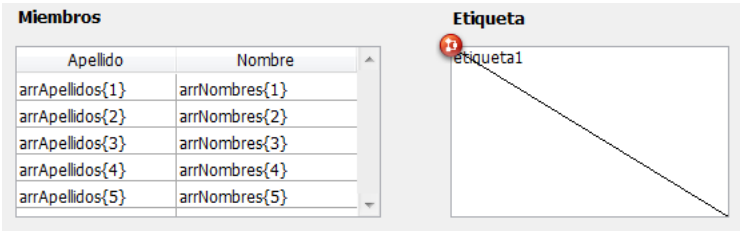
```
            :(($vlSrcType=String array) | ($vlSrcType=Text array))
   ` Ok para los arrays cadena y texto
               $0:=0
            :(($vlSrcType=ls longint) | ($vlSrcType=ls real)
               If(ls a list($vpSrcObj->))
   ` OK para las listas jerárquicas
                  $0:=0
               End if
         End case
      End if

   ` Uso de este evento para efectuar realmente la acción de arrastrar y soltar
     :(Form event=On Drop)
      $vtDraggedData:=""
   ` Obtener la información sobre el objeto fuente de arrastrar y soltar
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
   ` Obtener el tipo de los datos arrastrados
      $vlSrcType:=Type($vpSrcObj->)
      Case of
   ` Si es un array
         :(($vlSrcType=String array) | ($vlSrcType=Text array))
            If($vlPID # Current process)
   ` Leer el elemento desde la instancia de la variable en el proceso fuente
               GET PROCESS VARIABLE($vlPID;$vpSrcObj->{$vlSrcElem};$vtDraggedData)
            Else
   ` Copiar el elemento del array
               $vtDraggedData:=$vpSrcObj->{$vlSrcElem}
            End if
   ` Si es una lista
         :(($vlSrcType=ls real) | ($vlSrcType=ls longint))
   ` Si es una lista de otro proceso
            If($vlPID # Current process)
   `Obtener la referencia de la lista en el otro proceso
               GET PROCESS VARIABLE($vlPID;$vpSrcObj->;$vlList)
            Else
               $vlList:=$vpSrcObj->
            End if
   ` Si la lista existe
            If(ls a list($vpSrcObj->))
   `Obtener el texto del elemento cuya posición se obtuvo
               GET LIST ITEM($vlList;$vlSrcElem;$vlItemRef;$vsItemText)
               $vtDraggedData:=$vsItemText
            End if
         Else
   ` Es una variable cadena o texto
            If($vlPID # Current process)
               GET PROCESS VARIABLE($vlPID;$vpSrcObj->;$vtDraggedData)
            Else
               $vtDraggedData:=$vpSrcObj->
            End if
      End case
   ` Si hay efectivamente a soltar (el objeto fuente puede estar vacío)
      If($vtDraggedData # "")
         $1->:=$1->+$vtDraggedData
      End if
End case
```

Una vez haya implementado este método de proyecto, puede utilizarlo de este forma:

```
   ` Método de objeto del campo de texto [anyTable]aTextField

Case of
   ` ...
   :(Form event=On Drag Over)
      $0:=Handle dropping to text area(Self)

   :(Form event=On Drop)
      Handle dropping to text area(Self)
   ` ...
End case
```

Ejemplo 3

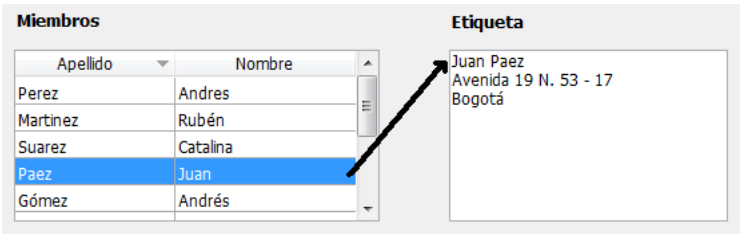
Queremos llenar un área de texto (por ejemplo, una etiqueta) con los datos arrastrados de una list box.



Este es el método de objeto de etiqueta1:

```
Case of
   :(Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
      If($source=Get pointer("listbox1"))
         $0:=0  `Se acepta soltar
      Else
         $0:=-1  `Se rechaza arrastrar
      End if
   :(Form event=On Drop)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
      QUERY([Empleados1];[Empleados1]Apellido=arrApellidos{$arrayrow})
      If(Records in selection([Empleados1])#0)
         etiqueta1:=[Empleados1]Nombre+" "+[Empleados1]Apellido+Char(Carriage return)+[Empleados1]Direccion+Char(Carriage return)+[Empleados1]Ciudad+","+" "+[Empleados1]Departamento+" "+[Empleados1]Codigopostal
      End if
End case
```

Luego se hace posible efectuar la siguiente acción:



Drop position

Drop position {{ colNum | posXImagen)} -> Resultado

Parámetro	Tipo		Descripción
colNum | posXImagen	Entero largo	⬅	Número de columna del list box (-1 si el soltar ocurre más allá de la última columna)o Posición coordenada X en la imagen
Resultado	Entero largo	➲	Número (array/ list box) o posición (lista jerárquica) o Posición en cadena (texto/combo box) o elemento de destino o -1 si soltar ocurrió más allá del último elemento del array o de la lista

Descripción

El comando **Drop position** puede utilizarse para descubrir la ubicación, en un objeto de destino "complejo", donde un objeto ha sido (arrastrado y) soltado.

Generalmente, se utiliza **Drop position** cuando administra un evento arrastrar y soltar que se produce en un array, un list box, una lista jerárquica, un campo de texto o una imagen.

- Si el objeto de destino es un array, el comando devuelve un número de elemento.
- Si el objeto de destino es un list box, el comando devuelve un número de línea. En este caso, el comando también devuelve el número de columna donde se soltó en el parámetro opcional *colNum*.
- Si el objeto de destino es una lista jerárquica, el comando devuelve una posición del elemento.
- Si el objeto de destino es una variable o un campo tipo texto, o un combo box, el comando devuelve una posición de carácter al interior de la cadena.
  En todos los casos, el comando puede devolver -1 si el objeto fuente ha sido soltado más allá del último elemento o del último elemento del objeto de destino.
- Si el objeto de destino es una variable o un campo de tipo imagen, la función devuelve la ubicación horizontal del clic y en el parámetro opcional *posYImagen*, la ubicación vertical del clic. Los valores devueltos se expresan en píxeles y con relación al sistema de coordenadas locales.

Si llama **Drop position** cuando procesa un evento que no es del tipo arrastrar y soltar en un array, un list box, un combo box, una lista jerárquica, un texto o una imagen, el comando devuelve -1.

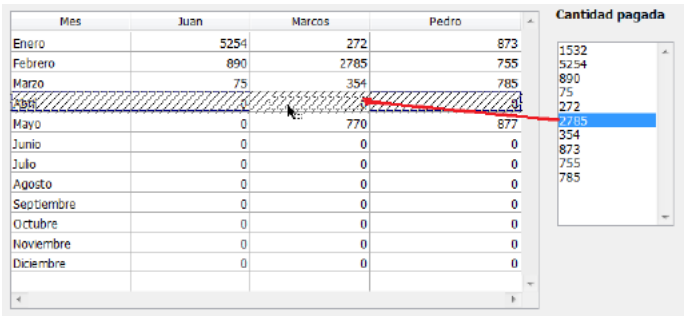
Importante: para que un objeto de formulario acepte los datos soltados, la propiedad **Soltable** debe estar seleccionada. Igualmente, su método de objeto debe ser activado por el evento On Drag Over y/o On Drop, para procesar estos eventos.

Ejemplo 1

Ver los ejemplos del comando **DRAG AND DROP PROPERTIES**.

Ejemplo 2

En el ejemplo siguiente, una lista de sumas debe ser desglosada por mes y por persona. La operación se efectúa arrastrando y soltando en un área de desplazamiento:



El método de objeto del list box contiene el siguiente código:

```
Case of
   :(Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($fuente;$arrayfila;$procesnum)
      If($fuente=Get pointer("SA1"))  `Si soltar proviene del área desplegable
         $0:=0
      Else
         $0:=-1  `Se rechaza soltar
      End if
   :(Form event=On Drop)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$procesnum)
      $filanum:=Drop position($colnum)
      If($colnum=1)
         BEEP
      Else
         Case of  `Adición de los valores soltados
            :($colnum=2)
```

```
                aJuan{$rownum}:=aJuan{$rownum}+SA1{$arrayfila}
            :($colnum=3)
                aMarcos{$rownum}:=aMarcos{$filanum}+SA1{$arrayfila}
            :($colnum=4)
                aPedro{$rownum}:=aPedro{$filanum}+SA1{$arrayfila}
        End case
        DELETE FROM ARRAY(SA1;$arrayfila)  `Área de actualización
    End if
End case
```

SET DRAG ICON

SET DRAG ICON (icono {; despH {; despV}})

Parámetro	Tipo		Descripción
icono	Imagen	➡	Icono a utilizar durante arrastrar
despH	Entero largo	➡	Desplazamiento horizontal del borde izquierdo de la imagen con respecto a la posición del cursor (> = 0, a la izquierda, <0 = a la derecha)
despV	Entero largo	➡	Desplazamiento vertical del borde superior de la imagen con respecto a la posición del cursor (> 0 = hacia arriba, <0 = hacia abajo)

Descripción

El comando **SET DRAG ICON** asocia la imagen de icono al cursor durante las operaciones de arrastrar y soltar que se manejan por programación.

Este comando sólo se puede llamar en el contexto del evento formulario On Begin Drag Over (ver el comando **Evento formulario**).

En el parámetro *icono*, pase la imagen que desea utilizar. Su tamaño máximo es de 256x256 píxeles. Si una de sus dimensiones excede los 256 píxeles, se redimensiona automáticamente.

En *despH* y *despV*, puede pasar valores de desplazamiento en píxeles:

- pase en *despH*, el desplazamiento horizontal del borde izquierdo del icono con respecto a la posición del cursor. Pase un valor positivo para aplicar este desplazamiento hacia la izquierda o hacia un valor negativo para aplicarlo a la derecha.
- pase en *despV*, el desplazamiento vertical desde el borde superior del icono con respecto a la posición del cursor. Pase un valor positivo para aplicar este desplazamiento hacia arriba o hacia un valor negativo para aplicarlo hacia abajo.

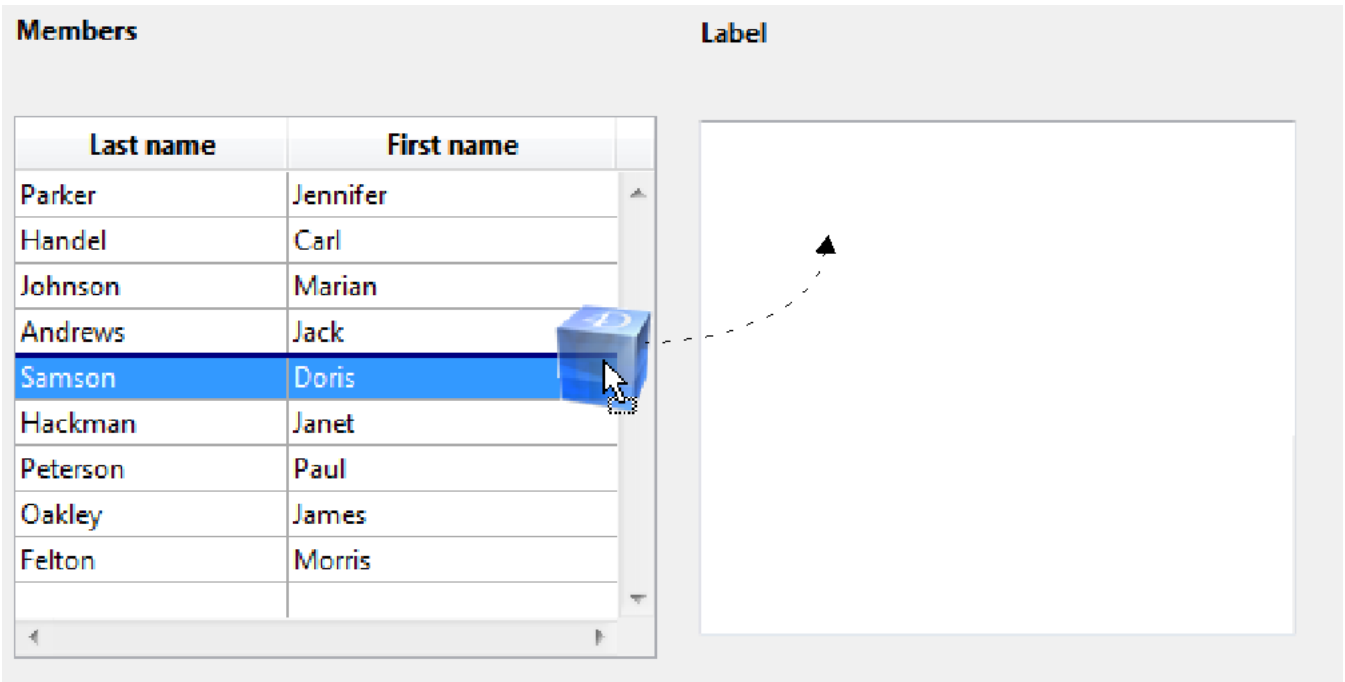
Si omite este parámetro, el cursor se coloca en el centro del icono.

Ejemplo

En un formulario, un usuario puede generar una etiqueta por arrastrar y soltar una fila. En el método objeto del lista box, puede escribir:

```
If(Form event=On Begin Drag Over)
   READ PICTURE FILE(Get 4D folder(Current resources folder)+"splash.png";vpict)
   CREATE THUMBNAIL(vpict;vpict;48;48)
   SET DRAG ICON(vpict)
End if
```

Al arrastrar una fila, la imagen aparecerá como se muestra aquí:



Note que puede modificar la posición del cursor con respecto a la imagen:

```
SET DRAG ICON(vpict;0;0)
```

Arrays

- Arrays
- Creación de arrays
- Arrays y objetos de formulario
- Arrays y el lenguaje 4D
- Arrays y punteros
- Utilizar el elemento cero de un array
- Arrays de dos dimensiones
- Arrays y memoria
- APPEND TO ARRAY
- ARRAY BLOB
- ARRAY BOOLEAN
- ARRAY DATE
- ARRAY INTEGER
- ARRAY LONGINT
- ARRAY OBJECT
- ARRAY PICTURE
- ARRAY POINTER
- ARRAY REAL
- ARRAY TEXT
- ARRAY TIME
- ARRAY TO LIST
- ARRAY TO SELECTION
- BOOLEAN ARRAY FROM SET
- COPY ARRAY
- Count in array
- DELETE FROM ARRAY
- DISTINCT ATTRIBUTE PATHS
- DISTINCT ATTRIBUTE VALUES
- DISTINCT VALUES
- Find in array
- Find in sorted array
- INSERT IN ARRAY
- LIST TO ARRAY
- LONGINT ARRAY FROM SELECTION
- MULTI SORT ARRAY
- SELECTION RANGE TO ARRAY
- SELECTION TO ARRAY
- Size of array
- SORT ARRAY
- TEXT TO ARRAY
- *_o_ARRAY STRING*

Arrays

Un **array** es una serie ordenada de variables del mismo tipo. Cada variable se llama un **elemento** del array. El **tamaño** de un array es el número de elementos que contiene. El tamaño del array se define al momento de su creación; luego puede redimensionarlo tantas veces como sea necesario añadiendo, insertando, o eliminando elementos, o redimensionando el array utilizando el mismo comando que utilizó para crearlo.

Se crea un array con uno de los comandos de declaración de Arrays. Para mayor información, consulte la sección **Creación de arrays**.

Los elementos se numeran de **1 a N**, donde N es el tamaño del Array. Un array siempre tiene un **elemento cero** al cual puede acceder tal como accede a cualquier otro elemento del array, pero este elemento no aparece cuando un Array se presenta en un formulario. Aunque el elemento cero no aparece cuando un array soporta un objeto de formulario, no hay restricción de utilizarlo con el lenguaje. Para mayor información sobre el elemento cero, consulte la sección **Utilizar el elemento cero de un array**.

Los arrays son variables 4D. Como toda variable, un Array tiene un alcance y sigue las reglas del lenguaje 4D, sólo con algunas diferencias únicas. Para mayor información, consulte las secciones **Arrays y el lenguaje 4D** y **Arrays y punteros**.

Los arrays son objetos del lenguaje; puede crear y utilizar arrays que nunca aparecerán en la pantalla. Los arrays también son objetos de la interfaz del usuario. Para mayor información sobre la interacción entre arrays y objetos de formulario, consulte la sección **Arrays y objetos de formularioÁreas de desplazamiento agrupadas**.

Los arrays están diseñados para manipular una cantidad razonable de datos por un corto período de tiempo. Sin embargo, como los arrays residen en memoria, son de fácil y rápida utilización. Para mayor información, consulte la sección **Arrays y memoria**.

Creación de arrays

Un array se crea con uno de los comandos de declaración de arrays descritos en este capítulo. Esta es la lista de comandos de declaración de arrays:

Comando	Crea o redimensiona un array de
ARRAY INTEGER	Enteros 2 bytes
ARRAY LONGINT	Enteros 4 bytes
ARRAY REAL	Reales
ARRAY TEXT	Textos (hasta 2 GB de texto por elemento) (*)
_o_ARRAY STRING	Textos (obsoleto) (*)
ARRAY DATE	Fechas
ARRAY BOOLEAN	Booleanos
ARRAY PICTURE	Imágenes
ARRAY POINTER	Punteros
ARRAY OBJECT	Objetos de lenguaje
ARRAY BLOB	BLOBs
ARRAY TIME	Horas

Cada comando de declaración de arrays puede crear o redimensionar arrays de una o dos dimensiones. Para mayor información sobre arrays de dos dimensiones, consulte la sección **Arrays de dos dimensiones**.

(*) No hay diferencia entre los arrays de tipo y los arrays Alfa. El parámetro *longCadena* del comando **_o_ARRAY STRING** se ignora. Se recomienda utilizar arrays Texto. El comando **_o_ARRAY STRING** se conserva por motivos de compatibilidad únicamente.

La siguiente línea de código crea (declara) un array de enteros de 10 elementos:

```
ARRAY INTEGER(aiAnArray;10)
```

Luego, el siguiente código redimensiona el mismo array a 20 elementos:

```
ARRAY INTEGER(aiAnArray;20)
```

Finalmente, el siguiente código redimensiona el mismo array a 0 elementos:

```
ARRAY INTEGER(aiAnArray;0)
```

Los elementos en un array se referencian utilizando llaves ({…}). Se utiliza un número dentro de las llaves para referirse a un elemento en particular; este número se llama número de elemento. Las siguientes líneas colocan cinco nombres en un array llamado *atNombres* y luego los muestra en ventanas de alerta:

```
ARRAY TEXT(atNombres;5)
atNombres{1}:="Ricardo"
atNombres{2}:="Sara"
atNombres{3}:="Samuel"
atNombres{4}:="Javier"
atNombres{5}:="Juan"
For($vlElem;1;5)
   ALERT("El elemento #"+String($vlElem)+" es igual a: "+atNombres{$vlElem})
End for
```

Observe la sintaxis *atNombres{$vlElem}.* En lugar de especificar un número literal tal como *atNombres{3}*, puede utilizar una variable numérica para indicar que elemento de array al que usted está direccionando.

Utilizando las iteraciones de las estructuras de bucle (**For...End for**, **Repeat...Until** o **While...End whileIs Windows**), pedazos compactos de código pueden direccionar todos o parte de los elementos en un Array.

Arrays y otras áreas del lenguaje 4D

Hay otros comandos 4D que pueden crear y trabajar con arrays. En particular:

- Para trabajar con arrays y selecciones de registros, utilice los comandos **SELECTION RANGE TO ARRAY**, **SELECTION TO ARRAY**, **ARRAY TO SELECTION** y **DISTINCT VALUES**.
- Los objetos de tipo List box están basados en arrays; varios comandos del tema “List box” trabajan con arrays, por ejemplo **LISTBOX INSERT ROWS**.
- Puede crear gráficos a partir de los valores almacenados en las tablas, subtablas y arrays. Para mayor información, consulte el comando **GRAPH**.

- Los comandos **LIST TO ARRAY** y **ARRAY TO LIST**.
- Muchos comandos pueden construir arrays en una llamada, por ejemplo: **FONT LIST**, **WINDOW LIST**, **VOLUME LIST**, **FOLDER LIST**, **DOCUMENT LIST**, **GET SERIAL PORT MAPPING**, **SAX GET XML ELEMENT**, etc.

Arrays y objetos de formulario

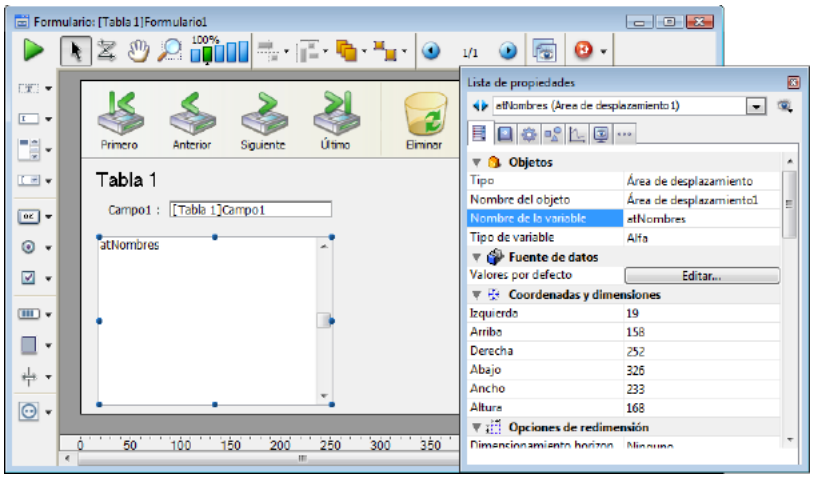
Los arrays son objetos del lenguaje, usted puede crear y utilizar arrays que nunca aparecerán en una pantalla. Sin embargo, los arrays también son objetos de la interfaz del usuario. Los siguientes tipos de **Objetos de formulario** están soportados por arrays:

- Menú desplegable
- Combo Box
- Área de desplazamiento (obsoleto a partir de 4D v13)
- Pestaña
- List box

Puede predefinir estos objetos en el editor de formularios del entorno Diseño utilizando el botón de los valores por defecto de la ventana Lista de propiedades (excepto List box), también puede definirlos por programación utilizando los comandos de arrays. En ambos casos, el objeto de formulario está soportado por un array creado por usted o por 4D.

Cuando utiliza estos objetos, puede detectar que elemento del objeto ha sido seleccionado (o recibió un clic) sometiendo a una prueba al **elemento seleccionado** del array. Inversamente, puede seleccionar un elemento particular del objeto designando el elemento seleccionado para el array.

Cuando un array se utiliza para generar un objeto de formulario, tiene una naturaleza doble; es a la vez un objeto del lenguaje y un objeto de la interfaz del usuario. Por ejemplo, cuando diseña un formulario y crea un área de desplazamiento:



El nombre de la variable asociada, en este caso *atNombres*, es el nombre del array que usted utiliza para crear y administrar el área de desplazamiento.

Notas:

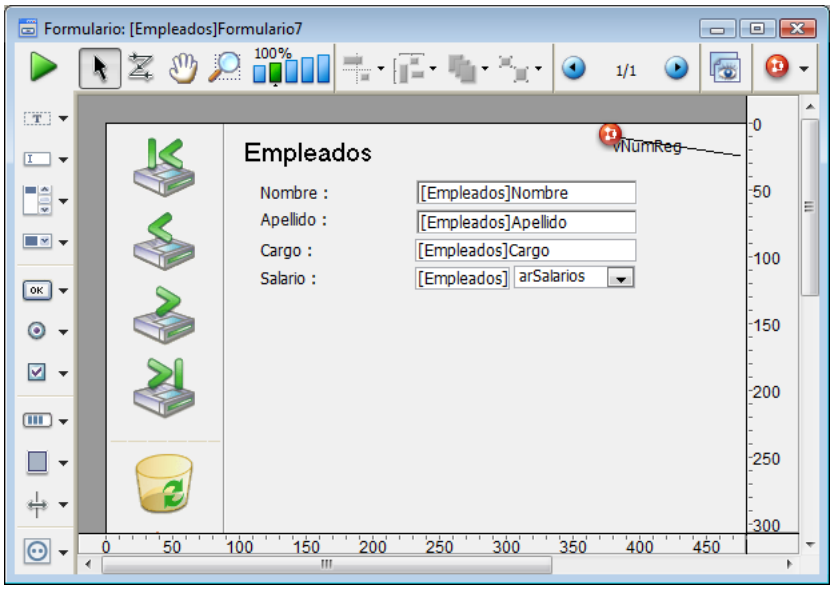
- El **elemento seleccionado** del array se almacena internamente en una variable de tipo entero. Como resultado, no es posible utilizar arrays que contengan más de 32.767 elementos (sin embargo, un gran número de elementos no es adecuado para la visualización como un objeto de formulario).
- No puede mostrar arrays de dos dimensiones ni arrays de punteros.
- La gestión de objetos de tipo **List box** (los cuales pueden contener varios arrays) implica numerosos aspectos específicos. Estas particularidades se tratan en la sección **Gestión programada de los objetos de tipo List box**.

Ejemplo: Creación de un menú desplegable

El siguiente ejemplo muestra cómo llenar un array y mostrarlo en una lista desplegable. Un array *arSalarios* se crea utilizando el comando **ARRAY REAL**. Contiene todos los salarios de las personas de una empresa. Cuando el usuario selecciona un elemento en el menú desplegable, el campo *[Empleados]Salario* recibe el valor escogido.

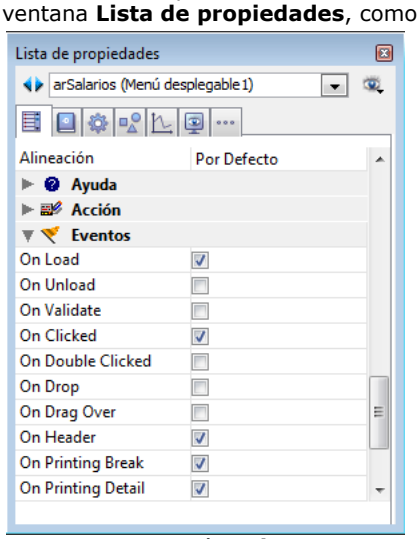
Creación del menu desplegable Salarios en un formulario

Cree un menú desplegable y llámela *arSalarios*. El nombre del menú desplegable debe ser el mismo del Array.

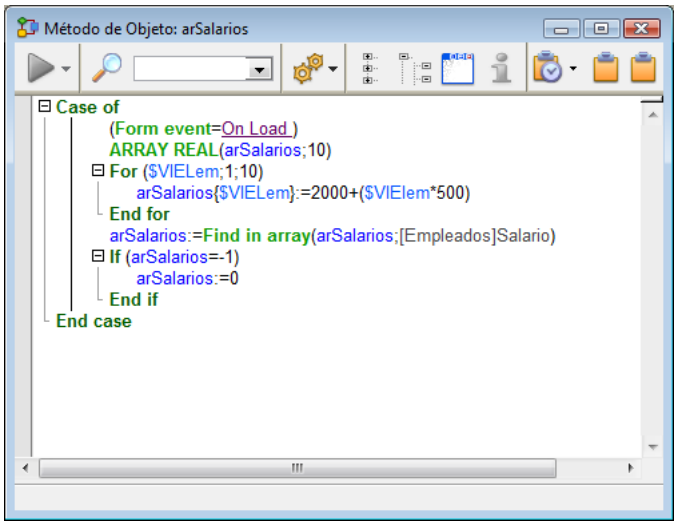


Inicialización del Array

Inicialice el Array *arSalarios* utilizando el evento **On Load** para el objeto. Para hacer esto, recuerde activar el evento en la ventana **Lista de propiedades**, como se muestra a continuación:



Haga clic en el botón **Método de objeto** y cree el siguiente método:



Las líneas:

```
ARRAY REAL(arSalarios;10)
For($vlElem;1;10)
    arSalarios{$vlElem}:=2000+($vlElem*500)
End for
```

crean el array numérico 2500, 3000... 7000, correspondiente a los salarios anuales de $30 000 a $84 000, antes de impuestos.
Las líneas:

```
arSalarios:=Find in array(arSalarios;[Empleados]Salario)
If(arSalarios=-1)
```

```
    arSalarios:=0
End if
```

manejan la creación de un nuevo registro o la modificación de un registro existente.

- Si crea un nuevo registro, el campo *[Empleados]Salario* inicialmente es igual a cero. En este caso, **Find in Array** no encuentra el valor en el array y devuelve -1. La prueba **If (arSalarios=-1)** coloca en cero a *arSalarios*, indicando que no se ha seleccionado un elemento en la lista desplegable.
- Si modifica un registro existente, **Find in Array** recupera el valor en el array y da al elemento seleccionado de la lista desplegable el valor actual del campo. Si el valor para un empleado no está en la lista, la prueba **If (arSalarios=-1)** deselecciona todos los elementos de la lista.

Nota: para mayor información sobre el **elemento de array seleccionado**, lea la próxima sección.

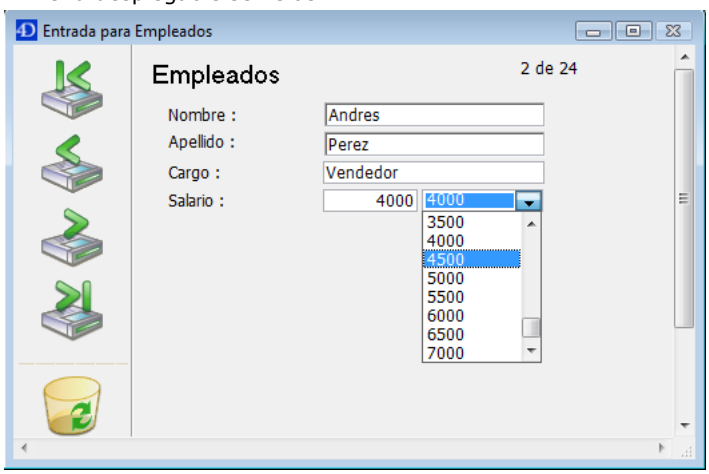
Asignación del valor seleccionado al campo [Empleados]Salario

Para reportar el valor seleccionado del menú desplegable *arSalarios,* sólo necesita administrar el evento **On Clicked** del objeto. El número de elemento seleccionado es el valor del array *arSalarios*. Por lo tanto, la expresión *arSalarios{arSalarios}* devuelve el valor seleccionado en la lista desplegable.

Complete el método de objeto *arSalarios* de esta forma:

```
Case of
   :(Form event=On Load)
      ARRAY REAL(arSalarios;10)
      For($vlElem;1;10)
         arSalarios{$vlElem}:=2000+($vlElem*500)
      End for
      arSalarios:=Find in array(arSalarios;[Empleados]Salario)
      If(arSalarios=-1)
         arSalarios:=0
      End if
   :(Form event=On Clicked)
      [Empleados]Salario:=arSalarios{arSalarios}
End case
```

El menú desplegable se ve así:



La siguiente sección describe las operaciones comunes y básicas que usted realizará sobre arrays cuando los utiliza como objetos de formulario.

Obtener el tamaño de un array

Puede obtener el tamaño actual de un array utilizando el comando **Size of Array**. Utilizando el ejemplo anterior, la siguiente línea de código mostrará 5:

```
ALERT("El tamaño del array atNombres es: "+String(Size of array(atNombres)))
```

Reordenar los elementos del array

Puede reordenar los elementos del array utilizando el comando **SORT ARRAY** o de diferentes arrays utilizando el comando **MULTI SORT ARRAY**. Utilizando el ejemplo anterior, y siempre que el array se muestre como un área de desplazamiento:

a. Al principio, el área se verá como la lista a la izquierda.

b. Después de la ejecución de la siguiente línea de código:

```
SORT ARRAY(atNombres;>)
```

el área se verá como la lista en el medio.

c. Después de la ejecución de la siguiente línea de código:

```
SORT ARRAY(atNombres;<)
```

el área se verá como la lista a la derecha.

Añadir o borrar elementos

Puede añadir, insertar, o borrar elementos utilizando los comandos **APPEND TO ARRAY**, **INSERT IN ARRAY** y **DELETE FROM ARRAY**.

Gestión de los clics en el array: probar el elemento seleccionado

Utilizando el ejemplo anterior, y siempre que el array se muestre como un área de desplazamiento, usted puede manejar los clics en esta área de la siguiente forma:

```
  ` Método de objeto área de desplazamiento atNombres
Case of
   :(Form event=On Load)
  ` Inicializar el array (como se mostró anteriormente)
      ARRAY TEXT(atNames;5)
  ` ...
   :(Form event=On Unload)
  ` No necesitamos más el array
      CLEAR VARIABLE(atNombres)

   :(Form event=On Clicked)
      If(atNombres#0)
         vtInfo:="Usted hace clic en: "+atNombres{atNombres}
      End if
   :(Form event=On Double Clicked)
      If(atNombres#0)
         ALERT("Usted hace doble clic en: "+atNombres{atNombres}
      End if
End case
```

Nota: los eventos deben ser activados en las propiedades del objeto.

Mientras la sintaxis *atNombres{$vlElem}* le permite trabajar con un elemento particular del array, la sintaxis *atNombres* devuelve el número del elemento seleccionado en el array. De esta manera, la sintaxis *atNombres{atNombres}* significa "el valor del elemento seleccionado en el array *atNombres.*" Si ningún elemento ha sido seleccionado, *atNombres* es igual a 0 (cero), de manera que la prueba **If (atNames#0)** detecta si un elemento ha sido seleccionado realmente o no.

Designar el elemento seleccionado

Puede cambiar por programación el elemento seleccionado asignando un valor al array.

Ejemplos

```
  ` Seleccionar el primer elemento (si el array no está vacío)
atNombres:=1

  ` Seleccionar el último elemento (si el array no está vacío)
atNombres:=Size of array(atNombres)

  ` Deseleccionar el elemento seleccionado (si lo hay) entonces ningún elemento está seleccionado
atNombres:=0

If((0<atNombres)&(atNombres<Size of array(atNombres))
  ` Si es posible, seleccionar el elemento siguiente al elemento seleccionado
   atNombres:=atNombres+1
End if

If(1<atNombres)
  ` Si es posible, seleccionar el elemento anterior al elemento seleccionado
```

```
    atNombres:=atNombres-1
End if
```

Búsqueda de un valor en el array

El comando **Find in Array** busca un valor particular en un array. Utilizando el ejemplo anterior, el siguiente código seleccionará el elemento cuyo valor es "Ricardo," si ese es el nombre introducido en la caja de diálogo de solicitud:

```
$vsNombre:=Request("Introduzca el nombre:")
If(OK=1)
    $vlElem:=Find in array(atNombres;$vsNombre)
    If($vlElem>0)
        atNombres:=$vlElem
    Else
        ALERT("No está "+$vsNombre+" en esta lista de nombres.")
    End if
End if
```

Los menús desplegables, áreas de desplazamiento y pestañas generalmente pueden ser administrados de la misma manera. Evidentemente, no es necesario código adicional para redibujar objetos en la pantalla cada vez que cambie el valor de un elemento, o añada o borre elementos.

Nota: para crear y utilizar pestañas con iconos y activar y desactivar pestañas, debe utilizar una lista jerárquica como objeto de soporte para la pestaña. Para mayor información, vea el ejemplo del comando **New list**.

Gestión de combo boxes

Mientras usted puede manejar menús desplegables, áreas de desplazamiento, y pestañas con los algoritmos descritos en la sección anterior, los combo boxes se manejan de forma diferente.

Un combo box es en realidad un área de entrada de texto la cual está asociada a una lista de valores (los elementos del array). El usuario puede elegir un valor de esta lista, y luego editar el texto. Por lo tanto, para un combo box, la noción de elemento seleccionado no aplica.

Con los combo boxes, nunca hay un elemento seleccionado. Cada vez que el usuario selecciona uno de los valores asociado al área, el valor se coloca en el elemento cero del array. Luego, si el usuario edita el texto, el valor modificado por el usuario también se coloca en el elemento cero.

Ejemplo

```
  ` Método de objeto Combo Box asColores
Case of
    :(Form event=On Load)
        ARRAY STRING(31;asColores;3)
        asColores{1}:="Azul"
        asColores{2}:="Blanco"
        asColores{3}:="Rojo"
    :(Form event=On Clicked)
        If(asColores{0}#"")
  ` El objeto cambia automáticamente su valor
  ` La utilización dle evento On Clicked con un Combo Box
  ` se necesita sólo cuando se deben tomar acciones adicionales
        End if
    :(Form event=On Data Change)
  ` Buscar en array ignora el elemento 0, de manera que devuelve -1 o >0
        If(Find in array(asColores;asColores{0})<0)
  ` El valor introducido no es uno de los valores adjuntos al objeto
  ` Añadir el valor a la lista para la próxima vez
            APPEND TO ARRAY(asColores;asColores{0})
        Else
  ` El valor introducido está entre los valores adjuntos al objeto
        End if
End case
```

Arrays y el lenguaje 4D

Los arrays son variables 4D. Como toda variable, un array tiene un alcance y sigue las reglas del lenguaje 4D language, aunque con algunas diferencias.

Arrays locales, proceso e interproceso

Puede crear y trabajar con arrays locales, proceso e interproceso. Ejemplos:

```
ARRAY INTEGER($aiCodigos;100)  ` Crea un array local de 100 valores enteros de 2 bytes
ARRAY INTEGER(aiCodigos;100)  ` Crea un array de proceso de 100 valores enteros de 2 bytes
ARRAY INTEGER(◊aiCodigos;100)  ` Crea un array interproceso de 100 valores enteros de 2 bytes
```

El alcance de estos arrays es idéntico al alcance de otras variables locales, proceso e interproceso:

Arrays locales

Se declara un array local cuando el nombre del array comienza con el signo dólar (*$*).

El alcance de un array local es el método en el que se crea. El array se borra cuando el método termina. Los arrays locales con el mismo nombre en dos métodos diferentes pueden tener diferentes tipos, porque en realidad son dos variables diferentes con diferentes alcances.

Cuando crea un array local en un método de formulario, de objeto o de proyecto llamado como subrutina por los dos tipos de métodos anteriores, el array se crea y se borra cada vez que se llama el formulario o el método de objeto. En otras palabras, el array se crea y borra para cada evento de formulario. Por lo tanto, no puede utilizar arrays locales en formularios, ni para visualizar ni para imprimir.

Como en el caso de las variables locales, es buena idea utilizar arrays locales cada vez que sea posible. Al hacerlo, minimiza la cantidad de memoria necesaria para correr su aplicación.

Arrays de proceso

Se declara un array de proceso cuando el nombre del array comienza con una letra.

El alcance de un array de proceso es el proceso en el que se crea. El array se borra cuando el proceso termina o es abortado. Un array de proceso dispone de una instancia creada automáticamente para cada proceso. Por lo tanto, el array es del mismo tipo para todos los procesos. Sin embargo, su contenido es particular para cada proceso.

Arrays interproceso

Se declara un array interproceso cuando el nombre del array comienza con <> (en Windows y Macintosh) o con el signo diamante, Opción-Mayús-V en un teclado en castellano (sólo en Macintosh).

El alcance de un array interproceso es la totalidad de procesos durante una sesión de trabajo. Deben ser utilizados sólo para compartir datos y transferir información entre procesos.

Truco: cuando usted sabe con anterioridad que un array interproceso será utilizado por varios procesos, lo cual puede provocar conflictos, proteja el acceso a este array con un semáforo. Para mayor información, vea el ejemplo para el comando **Semaphore**.

Nota: puede utilizar arrays de proceso e interproceso en formularios para crear objetos de formulario tales como áreas de desplazamiento, menús desplegables, etc.

Pasar un array como parámetro

Puede pasar un array como parámetro a un comando 4D o a una rutina de un plug-in 4D. Por otra parte, no puede pasar un array como parámetro para un método de usuario. La solución es pasar un puntero al array como parámetro al método. Para más detalles, consulte la sección **Arrays y punteros**.

Asignar un array a otro array

A diferencia de las variables de tipo texto o alfa, usted no puede asignar un array a otro. Para copiar (asignar) un array a otro, utilice la función **COPY ARRAY**.

Arrays y punteros

Puede pasar un array como parámetro a un comando 4D o a la rutina de un plug-in 4D. Por el contrario, no puede pasar un array como parámetro a un método de usuario. La alternativa es pasar un puntero al array como parámetro del método.

Nota: puede pasar arrays de proceso e interproceso como parámetros, pero no arrays locales.

Estos son algunos ejemplos.

- Miremos este ejemplo:

```
If((0<atNombres)&(atNombres<Size of array(atNombres))
   ` Si es posible, seleccione el elemento siguiente al elemento seleccionado
   atNombres:=atNombres+1
End if
```

Si necesita hacer lo mismo para 50 arrays diferentes, puede evitar escribir lo mismo 50 veces, utilizando el siguiente método de proyecto:

```
   ` Método de proyecto SELECCIONAR SIGUIENTE ELEMENTO
   ` (Puntero) SELECCIONAR SIGUIENTE ELEMENTO
   ` ( -> Array ) SELECCIONAR SIGUIENTE ELEMENTO

   C_POINTER($1)

   If((0<$1->)&($1-><Size of array($1->))
      $1->:=$1->+1  ` Si es posible, seleccione el elemento siguiente del elemento seleccionado
   End if
```

Luego, puede escribir:

```
SELECCIONAR SIGUIENTE ELEMENTO(->atNombres)
   ` ...
SELECCIONAR SIGUIENTE ELEMENTO(->asCodigosPostales)
   ` ...
SELECCIONAR SIGUIENTE ELEMENTO(->ailDRegistros)
   ` ... y así sucesivamente
```

- El siguiente método de proyecto devuelve la suma de todos los elementos de un array numérico (Entero, Entero largo, o real):

```
   ` Suma Array
   ` Suma Array ( Puntero)
   ` Suma Array ( -> Array )

   C_REAL($0)

   $0:=0
   For($vlElem;1;Size of array($1->))
      $0:=$0+$1->{$vlElem}
   End for
```

Nota: desde 4D v13, puede utilizar la función **Sum** para efectuar la suma de los elementos de un array numérico.

Luego, puede escribir:

```
$vlSum:=Suma Array(->arSalarios)
   ` ...
$vlSum:=Suma Array(->aiDefectCounts)
   ` ..
$vlSum:=Suma Array(->alPoblacion)
```

• El siguiente método de proyecto pasa a mayúsculas todos los elementos de un array alfa o texto:

```
   ` MAYUSCULAS ARRAY
   ` MAYUSCULAS ARRAY ( Puntero )
   ` MAYUSCULAS ARRAY ( -> Array )
```

```
For($vlElem;1;Size of array($1->))
    If($1->{$vlElem}#"")
        $1->{$vlElem}:=Uppercase($1->{$vlElem}[[1]])+Lowercase(Substring($1->{$vlElem};2))
    End if
End for
```

Luego, puede escribir:

```
MAYUSCULAS ARRAY(->atTemas)
  ` ...
MAYUSCULAS ARRAY(->asApellidos)
```

La combinación de arrays, punterors, y estructuras de bucles, tales como **For... End for,** le permite escribir muchos métodos de proyecto útiles para manejar arrays.

Utilizar el elemento cero de un array

Un array siempre tiene un elemento cero. Mientras el elemento cero no se muestra cuando un array maneja un objeto de formulario, no hay restricción(*) en utilizarlo con el lenguaje.

Un ejemplo del uso del elemento cero es el caso del combo box de la sección **Arrays y objetos de formulario**.

Otros ejemplo: si quiere ejecutar una acción sólo cuando haga clic en un elemento diferente del previamente seleccionado, debe hacer seguimiento de cada elemento seleccionado. Una manera de hacer esto es utilizar una variable de proceso en la cual usted mantiene el número del elemento seleccionado. Otra forma es utilizar el elemento cero del array:

```
  ` Método de objeto área de desplazamiento atNombres
Case of
   :(Form event=On Load)
  ` Inicialización del array (como se mostró anteriormente)
      ARRAY TEXT(atNombres;5)
  ` ...
  ` Inicializar el elemento cero con el número
  ` de elemento actual seleccionado en su forma alfanúmerica
  ` Acá usted comienza sin elemento seleccionado
      atNombres{0}:="0"

   :(Form event=On Unload)
  ` No necesitamos más el array
      CLEAR VARIABLE(atNames)

   :(Form event=On Clicked)
      If(atNames#0)
         If(atNames#Num(atNombres{0}))
            vtInfo:="Haga clic en: "+atNombres{atNombres}+" que no fue seleccionado anteriormente."
            atNombres{0}:=String(atNombres)
         End if
      End if
   :(Form event=On Double Clicked)
      If(atNombres#0)
         ALERT("Usted hace doble clic en: "+atNombres{atNombres}
      End if
End case
```

<gen9></gen9>

En este ejemplo avanzado, si un flujo de caracteres contiene caracteres **NULL** (código ASCII cero) se envía o recibe, el elemento cero de los arrays *<>aiMapaSalida* y *<>aiMapaEntrada* jugará su papel como otro elemento de los 255 elementos de los 255 arrays.

(*) Sin embargo, hay una excepción: en los **List Box** de tipo array, el elemento cero se utiliza internamente para guardar el valor anterior de un elemento en edición, de manera que no es posible utilizarlo en este contexto particular.

Arrays de dos dimensiones

Cada comando de declaración de array puede crear o redimensionar arrays de una o dos dimensiones. Ejemplo:

```
ARRAY TEXT(atTemas;100;50)  ` Crear un array texto compuesto de 100 filas de 50 columnas
```

Los arrays de dos dimensiones son esencialmente objetos del lenguaje; no es posible visualizarlos o imprimirlos.
En el ejemplo anterior:

- *atTemas* es un array de dos dimensiones
- *atTemas{8}{5}* es el elemento 5 (columna 5...) de la fila 8
- *atTemas{20}* es la fila 20 y un array de una dimensión
- **Size of Array(atTemas)** devuelve 100, que es el número de filas
- **Size of Array(atTemas{17})** devuelve 50, que es el número de columnas de la fila 17

En el siguiente ejemplo, un puntero a cada campo de cada tabla en la base se almacena en un array de dos dimensiones:

```
C_LONGINT($vlUltimaTabla;$vlUltimoCampo)
C_LONGINT($vlNumeroCampo)
  ` Crear tantas filas vacías como tablas
$vlUltimaTabla:=Get last table number
ARRAY POINTER(<>apCampos;$vlUltimaTabla;0)  `Array de dos dimensiones con X lineas y 0 columnas
  ` Para cada tabla
For($vlTabla;1;$vlUltimaTabla)
   If(Is table number valid($vlTabla))
      $vlUltimoCampo:=Get last field number($vlTabla)
  ` Definir los valores de los elementos
      $vlNumeroColumna:=0
      For($vlCampo;1;$vlUltimoCampo)
         If(Is field number valid($vlTabla;$vlCampo))
            $vlNumeroColumna:=$vlNumeroColumna+1
  `Inserte una columna en la línea de la tabla en uso
            INSERT IN ARRAY(<>apCampos{$vlTabla};$vlNumeroColumna;1)
  `Asigne la "celda" con el puntero
            <>apCampos{$vlTabla}{$vlNumeroColumna}:=Field($vlTabla;$vlCampo)
         End if
      End for
   End if
End for
```

En la medida en que este array de dos dimensiones haya sido inicializado, puede obtener los punteros a los campos para una tabla particular de la siguiente forma:

```
  ` Obtener los punteros a los campos para la tabla en pantalla::
COPY ARRAY(<>apCampos{Table(Current form table)};$apCamposActuales)
  ` Inicializar los campos booleanos y fecha
For($vlElemento;1;Size of array($apCamposActuales))
   Case of
      :(Type($apCamposActuales{$vlElemento}->)=Is date)
         $apCamposActuales{$vlElemento}->:=Current date
      :(Type($apCamposActuales{$vlElemento}->)=Is Boolean)
         $apCamposActuales{$vlElemento}->:=True
   End case
End for
```

Nota: como lo sugiere este ejemplo, las filas de un array de dos dimensiones pueden tener o no el mismo tamaño.

Arrays y memoria

A diferencia de los datos que usted almacena en disco utilizando tablas y registros, un array siempre está en memoria.

Por ejemplo, si todos los códigos postales de Estados Unidos fueran introducidos en la tabla *[Cogidos Postales]*, la tabla tendría alrededor de 100 000 registros. Además, esta tabla incluiría varios campos: el código postal y la ciudad, condado y estado correspondiente. Si selecciona sólo los códigos postales de California, 4D crea la selección de registros correspondiente al interior de la tabla *[Codigos Postales]*, y sólo carga los registros cuando sea necesario (por ejemplo, cuando se visualizan o imprimen). En otras palabras, usted trabaja con una serie ordenada de valores (del mismo tipo para cada campo) que se carga parcialmente del disco a la memoria por el motor de la base de 4D.

Hacer lo mismo con arrays sería muy complicado por las siguientes razones:

- Para mantener los cuatro tipos de información (código postal, ciudad, condado, estado), tendría que mantener cuatro arrays grandes en memoria.
- Como un array siempre está totalmente en memoria, tendría que mantener todos los códigos postales en memoria durante la sesión de trabajo, incluso si los datos no están siempre en uso.
- Nuevamente, como un array siempre está en memoria en su totalidad, cada vez que inicia y sale de la base, los cuatro arrays tendrían que cargarse y guardarse en el disco, aún cuando los datos no se utilicen o modifiquen durante la sesión de trabajo.

Conclusión: los arrays están pensados para manipular una cierta cantidad de datos por un periódo corto. Por otra parte, como los arrays están en memoria, se utilizan fácil y rápidamente.

Sin embargo, en algunas circusntancias, podría necesitar trabajar con arrays de cientos o miles de elementos. La siguiente tabla lista las fórmulas utilizadas para calcular la cantidad de memoria utilizada para cada tipo de array:

Tipo de array	**Fórmula para determinar la cantidad de memoria utilizada en bytes**
Booleano	(31+Número de elementos)\8
Fecha	(1+Número de elementos) * 6
Alfa (modo Unicode)	(1+Número de elementos) * (Suma del tamaño de cada texto)
Entero	(1+Número de elementos) * 2
Entero largo	(1+Número de elementos) * 4
Imagen	(1+Número de elementos) * 4 + Suma del tamaño de cada imagen
Puntero	(1+Número de elementos) * 16
Real	(1+Número de elementos) * 8
Texto (modo Unicode)	(1+Número de elementos) * 6 + Suma del tamaño de cada texto
Dos dimensiones	(1+Número de elementos) * 12 + Suma del tamaño de cada array

Notas:

- El tamaño de un texto en memoria se calcula utilizando la fórmula: ((Longitud + 1) * 2)
- Algunos pocos bytes adicionales son necesarios para mantener contacto con el elemento seleccionado, el número de elementos y el mismo array.

Cuando trabaja con arrays muy grandes, la mejor forma de manejar las situaciones de saturación de la memoria es acompañar la creación del array de un método de proyecto **ON ERR CALL**. Ejemplo:

```
  ` Usted va a ejecutar una operación por lotes toda la noche
  ` que requiere la creación de arrays grandes. Para evitar
  ` errores en la noche, cree los arrays al comienzo de la operación
  ` y pruebe los errores en este momento:
gError:=0  ` Inicialización
ON ERR CALL("MANEJO DE ERRORES")  ` Instalación del método de gestión de errores
ARRAY STRING(63;asEsteArray;50000)  ` Apróximadamente 3125K en modo ASCII
ARRAY REAL(arOtroArray;50000)  ` 488K
ON ERR CALL("")  `Ya no es necesario interceptar errores
If(gError=0)
  ` Los arrays pudieron ser creados
  ` y vamos a seguir con la operación
Else
   ALERT("Esta operación necesita más memoria")
End if
  ` Cualquiera que sea el caso, no necesitamos más los arrays
CLEAR VARIABLE(asEsteArray)
CLEAR VARIABLE(arOtroArray)
```

El método de proyecto ***MANEJO DE ERRORES*** es el siguiente:

```
  ` Método de proyecto MANEJO DE ERRORES
gError:=Error  ` Devolver el código del error
```

APPEND TO ARRAY

APPEND TO ARRAY (array ; valor)

Parámetro	Tipo		Descripción
array	Array	➡	Array al cual añadir un elemento
valor	Expresión	➡	Valor a añadir

Descripción

El comando **APPEND TO ARRAY** añade un nuevo elemento al final del *array* y asigna *valor* al elemento. En modo interpretado, si a*rray* no existe, el comando lo crea con respecto al tipo de *valor*.

Este comando funciona con todo tipo de arrays: cadena, numérico, booleano, fecha, puntero e imagen.

El tipo de *valor* debe corresponder al tipo de array, de lo contrario se genera el error de sintaxis 54 "Los tipos de argumentos son incompatibles". Los siguientes valores, sin embargo, se aceptan:
• un *array* de tipo cadena (Texto o Alfa) acepta todo *valor* de tipo Texto o Alfa.
• un *array de* de tipo numérico (Entero, Entero largo o Real) acepta todo *valor* de tipo Entero, Entero largo, Numérico u Hora.

Ejemplo

El siguiente código:

```
INSERT IN ARRAY($miarray;Size of array($miarray)+1)
$miarray{Size of array($miarray)}:=$mivalor
```

... puede reemplazarse por:

```
APPEND TO ARRAY($miarray;$mivalor)
```

ARRAY BLOB

ARRAY BLOB (nomArray ; tam {; tam2})

Parámetro	Tipo		Descripción
nomArray	Array	➡	Nombre del array
tam	Entero largo	➡	Número de elementos del array o número de arrays si se especifica tam2
tam2	Entero largo	➡	Número de los elementos de los array 2D

Descripción

El comando **ARRAY BLOB** crea y/o cambia el tamaño de un array de elementos de tipo Blob en memoria .

El parámetro *nomArray* es el nombre de la array.

El parámetro *tam* es el número de elementos del array.

El parámetro *tam2* es opcional. Si lo pasa, este comando crea un array de dos dimensiones. En este caso, *tam* especifica el número de filas y *tam2* el número de columnas de cada array. Cada fila en un array de dos dimensiones se puede procesar tanto como un elemento y como un array. Esto significa que cuando se trabaja con la primera dimensión de un array de dos dimensiones, puede insertar y retirar arrays enteros utilizando otros comandos en este tema.

Cuando se aplica el comando **ARRAY BLOB** a un array existente:

- Si amplía su tamaño, los elementos existentes no se cambian y los nuevos elementos se inicializan en un BLOB vacío (**BLOB size**= 0).
- Si reduce su tamaño, se eliminan y pierden los elementos de abajo del array.

Ejemplo 1

Este ejemplo crea un array proceso que contiene 100 elementos de tipo BLOB:

```
ARRAY BLOB(arrBlob;100)
```

Ejemplo 2

Este ejemplo crea un array local de 100 filas, que contienen cada una 50 elementos de tipo BLOB:

```
ARRAY BLOB($arrBlob;100;50)
```

Ejemplo 3

Este ejemplo crea un array local de 100 filas, conteniendo cada una 50 elementos de tipo BLOB. La variable $*vByteValue* recibe el décimo byte del BLOB ubicado en la séptima columna y la quinta fila del array BLOB:

```
C_INTEGER($vByteValue)
ARRAY BLOB($arrValues;100;50)
...
$vByteValue:=$arrValues{5}{7}{9}
```

ARRAY BOOLEAN

ARRAY BOOLEAN (nombreArray ; tamaño {; tamaño2})

Parámetro	Tipo		Descripción
nombreArray	Array	➡	Nombre del array
tamaño	Entero largo	➡	Número de elementos en el array o Número de filas si se especifica tamaño2
tamaño2	Entero largo	➡	Número de columnas en un array bidimensional

Descripción

El comando **ARRAY BOOLEAN** crea y/o redimensiona un array de elementos **Booleanos** en memoria.

- El parámetro *arrayName* es el nombre del array.
- El parámetro *tamaño* es el número de elementos en el array.
- El parámetro *tamaño2* es opcional; si se especifica *tamaño2*, el comando crea un array bidimensional.

En este caso, *tamaño* especifica el número de filas y *tamaño2* especifica el número de columnas en cada array. Cada fila en un array bidimensional puede tratarse como un elemento y como un array. Esto significa que mientras trabaja con la primera dimensión del array, puede utilizar otros comandos de array para insertar y borrar arrays enteros en un array bidimensional.

Cuando aplica **ARRAY BOOLEAN** a un array existente:

- Si agranda el tamaño del array, los elementos existente no son modificados, y los nuevos elementos se inicializan en False.
- Si reduce el tamaño del array, se pierden los últimos elementos borrados del array.

Tip: en algunos contextos, una alternativa a utilizar arrays booleanos es utilizar un array entero donde cada elemento signifique "verdadero" si es diferente de cero y signifique "falso" si es igual a cero.

Ejemplo 1

Este ejemplo crea un array de proceso de 100 elementos de tipo *Booleano*:

```
ARRAY BOOLEAN(abValores;100)
```

Ejemplo 2

Este ejemplo crea un array local de 100 filas de 50 elementos de tipo *Booleano*:

```
ARRAY BOOLEAN($abValores;100;50)
```

Ejemplo 3

Este ejemplo crea un array interproceso de 50 elementos de tipo *Booleano* y a cada elemento asigna el valor Verdadero par:

```
ARRAY BOOLEAN(◇abValues;50)
For($vlElem;1;50)
   ◇abValores{$vlElem}:=(($vlElem%2)=0)
End for
```

ARRAY DATE

ARRAY DATE (nombreArray ; tamaño {; tamaño2})

Parámetro	Tipo		Descripción
nombreArray	Array	➡	Nombre del array
tamaño	Entero largo	➡	Número de elementos en el array o Número de filas si se especifica tamaño2
tamaño2	Entero largo	➡	Número de columnas en un array bidimensional

Descripción

El comando **ARRAY DATE** crea y/o redimensiona un array de elementos de tipo Fecha en memoria.

- El parámetro *nombreArray* es el nombre del array.
- El parámetro *tamaño* es el número de elementos en el array.
- El parámetro *tamaño2* es opcional; si se especifica *tamaño2*, el comando crea un array bidimensional. En este caso, *tamaño* especifica el número de filas y *tamaño2* especifica el número de columnas en cada array. Cada fila en un array bidimensional puede tratarse como un elemento y como un array. Esto significa que mientras trabaja con la primera dimensión del array, puede utilizar otros comandos de array para insertar y borrar arrays enteros en un array bidimensional.

Cuando aplica **ARRAY DATE** a un array existente:

- Si agranda el tamaño del array, los elementos existente no son modificados, y los nuevos elementos se inicializan en (!00/00/00!).
- Si reduce el tamaño del array, se pierden los últimos elementos borrados del array.

Ejemplo 1

Este ejemplo crea un array de proceso de 100 elementos de tipo Fecha:

```
ARRAY DATE(adValores;100)
```

Ejemplo 2

Este ejemplo crea un array local de 100 filas de 50 elementos de tipo Fecha:

```
ARRAY DATE($adValores;100;50)
```

Ejemplo 3

Este ejemplo crea un array interproceso de 50 elementos de tipo Fecha y asigna a cada elemento la fecha actual más un número de días igual al número de elemento:

```
ARRAY DATE(◇adValores;50)
For($vlElem;1;50)
   ◇adValores{$vlElem}:=Current date+$vlElem
End for
```

ARRAY INTEGER

ARRAY INTEGER (nombreArray ; tamaño {; tamaño2})

Parámetro	Tipo		Descripción
nombreArray	Array	➡	Nombre del array
tamaño	Entero largo	➡	Nombre de los elementos en el array o Número de filas si se especifica el tamaño 2
tamaño2	Entero largo	➡	Número de columnas en un array de dos dimensiones

Descripción

El comando **ARRAY INTEGER** crea y/o redimensiona un array de elementos de tipo *Entero* de 2 bytes en memoria.

- El parámetro *nombreArray* es el nombre del array.
- El parámetro *tamaño* es el número de elementos en el array.
- El parámetro *tamaño2* es opcional; si especifica *tamaño2*, el comando crea un array de dos dimensiones. En este caso, *tamaño* especifica el número de filas y *tamaño2* el número de columnas en cada array. Cada fila en un array de dos dimensiones puede tratarse como un elemento y como un array. Esto significa que mientras trabaja con la primera dimensión del array, puede utilizar otros comando de array para insertar y borrar arrays enteros en un array de dos dimensiones.

Cuando aplica **ARRAY INTEGER** a un array existente:

- Si agranda el tamaño del array, los elementos existentes no son modificados, y los elementos nuevos se inicializan en 0.
- Si reduce el tamaño del array, los últimos elementos son borrados del array y se pierden.

Ejemplo 1

Este ejemplo crea un array proceso de 2 bytes de 100 elementos de tipo *Entero*:

```
ARRAY INTEGER(aiValores;100)
```

Ejemplo 2

Este ejemplo crea un array local de 2 bytes de 100 filas de 50 elementos de tipo *Entero*:

```
ARRAY INTEGER($aiValores;100;50)
```

Ejemplo 3

Este ejemplo crea un array interproceso de 2 bytes de 50 elementos de tipo *Entero*, y asigna a cada elemento su número de elemento:

```
ARRAY INTEGER(◇aiValores;50)
For($vlElem;1;50)
   ◇aiValores{$vlElem}:=$vlElem
End for
```

ARRAY LONGINT

ARRAY LONGINT (nombreArray ; tamaño {; tamaño2})

Parámetro	Tipo		Descripción
nombreArray	Array	➡	Nombre del array
tamaño	Entero largo	➡	Número de elementos en el array o Número de filas si se especifica tamaño2
tamaño2	Entero largo	➡	Número de columnas en un array de dos dimensiones

Descripción

El comando **ARRAY LONGINT** crea y/o redimensiona un array de elementos de tipo *Entero largo* de 4 bytes en memoria.

- El parámetro *nombreArray* es el nombre del Array.
- El parámetro *tamaño* es el número de elementos en el array.
- El parámetro *tamañoi2* es opcional; si se especifica *tamaño2*, el comando crea un array de dos dimensiones. En este caso, *tamaño* especifica el número de filas y *tamaño2* el número de columnas en cada array. Cada fila en un array de dos dimensiones puede tratarse como un elemento y un array. Esto significa que mientras trabaja con la primera dimensión del arrray, puede utilizar otros comandos de array para insertar y borrar arrays completos en un array de dos dimensiones.

Cuando aplica **ARRAY LONGINT** a un array existente:

- Si agranda el tamaño del array, los elementos existentes no son modificados, y los elementos nuevos se inicializan en 0.
- Si reduce el tamaño del array, los últimos elementos son borrados del array y se pierden.

Ejemplo 1

Este ejemplo crea un array proceso de 4 bytes de 100 elementos de tipo *Entero largo*:

```
ARRAY LONGINT(aiValores;100)
```

Ejemplo 2

Este ejemplo crea un array local de 4 bytes de 100 filas de 50 elementos de tipo *Entero largo*:

```
ARRAY LONGINT($aiValores;100;50)
```

Ejemplo 3

Este ejemplo crea un array interproceso de 4 bytes de 50 elementos de tipo *Entero largo* y asigna a cada elemento su número:

```
ARRAY LONGINT(◊aiValores;50)
For($vlElem;1;50)
   ◊aiValores{$vlElem}:=$vlElem
End for
```

ARRAY OBJECT

ARRAY OBJECT (nomArray ; tam {; tam2})

Parámetro	Tipo		Descripción
nomArray	Array	➡	Nombre del array
tam	Entero largo	➡	Número de elementos del array o número de arrays si se especifica tam2
tam2	Entero largo	➡	Número de elementos del array 2D

Descripción

El comando **ARRAY OBJECT** crea y/o redimensiona un array de elementos de tipo Objeto de lenguaje en memoria.
El parámetro *nomArray* es el nombre del array. Puede utilizar cualquier nombre conforme a las convenciones de 4D.

El parámetro de *tam* es el número de elementos del array.

El parámetro *tam2* es opcional. Si lo pasa, este comando crea un array de dos dimensiones. En este caso, *tam* especifica el número de filas y *tam2* el número de columnas de cada array. Cada fila en un array de dos dimensiones se puede procesar tanto como un elemento y como un array. Esto significa que cuando se trabaja con la primera dimensión de un array de dos dimensiones, se puede insertar y retirar arrays enteros utilizando otros comandos del tema "Arrays".

Cuando se aplica el comando **ARRAY OBJECT** a un array existente:

- Si amplía su tamaño, los elementos existentes no se cambian y los nuevos elementos no están definidos. Puede probar si un elemento se define utilizando el comando **OB Is defined**.
- Si reduce su tamaño, se eliminan y pierden los elementos al "fondo" del array.

Ejemplo 1

Creación de un array de proceso de 100 elementos de tipo objeto:

```
ARRAY OBJECT(arrObjects;100)
```

Ejemplo 2

Creación de un array local de 100 filas, conteniendo cada uno 50 elementos de tipo de objeto:

```
ARRAY OBJECT($arrObjects;100;50)
```

Ejemplo 3

Creación y llenado de un array local de objetos:

```
C_OBJECT($Children;$ref_richard;$ref_susan;$ref_james)
ARRAY OBJECT($arrayChildren;0)
OB SET($ref_richard;"name";"Richard";"age";7)
APPEND TO ARRAY($arrayChildren;$ref_richard)
OB SET($ref_susan;"name";"Susan";"age";4)
APPEND TO ARRAY($arrayChildren;$ref_susan)
OB SET($ref_james;"name";"James";"age";3)
APPEND TO ARRAY($arrayChildren;$ref_james)
 // $arrayChildren{1} -> {"name":"Richard","age":7}
 // $arrayChildren{2} -> {"name":"Susan","age":4}
 // $arrayChildren{3} -> {"name":"James","age":3}
```

ARRAY PICTURE

ARRAY PICTURE (nombreArray ; tamaño {; tamaño2})

Parámetro	Tipo		Descripción
nombreArray	Array	➡	Nombre del array
tamaño	Entero largo	➡	Número de elementos en el array o Número de filas si se especifica tamaño2
tamaño2	Entero largo	➡	Número de columnas en un array bidimensional

Descripción

El comando **ARRAY PICTURE** crea y/o redimensiona un array de elementos de tipo *Imagen* en memoria.

- El parámetro *nombreArray* es el nombre del array.
- El parámetro *tamaño* es el número de elementos en el array.
- El parámetro *tamaño2* es opcional; si se especifica *tamaño2*, el comando crea un array bidimensional. En este caso, *tamaño* especifica el número de filas y *tamaño2* especifica el número de columnas en cada array. Cada fila en un array bidimensional puede tratarse como un elemento y como un array. Esto significa que mientras trabaja con la primera dimensión del array, puede utilizar otros comandos de array para insertar y borrar arrays enteros en un array bidimensional.

Cuando aplica **ARRAY PICTURE** a un array existente:

- Si agranda el tamaño del array, los elementos existentes no son modificados, y los nuevos elementos se inicializan en imágenes vacías. Esto significa que **Picture size** aplicado a uno de estos elementos devuelve *0*.
- Si reduce el tamaño del array, se pierden los últimos elementos borrados del array.

Ejemplo 1

Este ejemplo crea un array de proceso de 100 elementos de tipo *Imagen*:

```
ARRAY PICTURE(agValores;100)
```

Ejemplo 2

Este ejemplo crea un array local de 100 filas de de 50 elementos de tipo *Imagen*:

```
ARRAY PICTURE($agValores;100;50)
```

Ejemplo 3

Este ejemplo crea un array interpcoceso de elementos de tipo *Imagen* y carga cada imagen en uno de los elementos del array. El tamaño del array es igual al número de recursos *'PICT'* disponibles en la base. El nombre del recurso del array comienza por *"User Intf/"*:

```
RESOURCE LIST("PICT";$aiResIDs;$asResNombres)
ARRAY PICTURE(◇agValores;Size of array($aiResIDs))
$vlPictElem:=0
For($vlElem;1;Size of array(◇agValues))
   If($asResNames="User Intf/@")
      $vlPictElem:=vlPictElem+1
      GET PICTURE RESOURCE("PICT";$aiResIDs{$vlElem};$vgImagen)
      ◇agValores{$vlPictElem}:=$vgImagen
   End if
End for
ARRAY PICTURE(◇agValores;$vlPictElem)
```

ARRAY POINTER

ARRAY POINTER (nombreArray ; tamaño {; tamaño2})

Parámetro	Tipo		Descripción
nombreArray	Array	➡	Nombre del array
tamaño	Entero largo	➡	Número de elementos en el array o Número de filas si se especifica tamaño2
tamaño2	Entero largo	➡	Número de columnas en un array bidimensional

Descripción

El comando **ARRAY POINTER** crea o redimensiona un array de elementos de tipo *Puntero* en memoria.

- El parámetro *nombreArray* es el nombre del array.
- El parámetro *tamaño* es el número de elementos en el array.
- El parámetro *tamaño2* es opcional; si se especifica *tamaño2*, el comando crea un array bidimensional. En este caso, *tamaño* especifica el número de filas y *tamaño2* especifica el número de columnas en cada array. Cada fila en un array bidimensional puede tratarse como un elemento y como un array. Esto significa que mientras trabaja con la primera dimensión del array, puede utilizar otros comandos de array para insertar y borrar arrays enteros en un array bidimensional.

Cuando aplica **ARRAY POINTER** a un array existente:

- Si agranda el tamaño del array, los elementos existentes no son modificados, y los nuevos elementos se inicializan en un *puntero* nulo. Esto significa queal aplicar **Nil** a uno de estos elementos devuelve Verdadero.
- Si reduce el tamaño del array, se pierden los últimos elementos borrados del array.

Ejemplo 1

Este ejemplo crea un array de proceso de 100 elementos de tipo *Puntero*:

```
ARRAY POINTER(apValores;100)
```

Ejemplo 2

Este ejemplo crea un array local de 100 filas de 50 elementos de tipo *Puntero*:

```
ARRAY POINTER($apValores;100;50)
```

Ejemplo 3

Este ejemplo crea un array interproceso de elementos de tipo *Puntero* y asigna a cada elemento al que apunta a la tabla cuyo número es el mismo del elemento. El tamaño del array es igual al número de tablas en la base de datos. En el caso de suprimir una tabla, la linea devolverá Nil

```
ARRAY POINTER(◇apValores;Get last table number)
For($vlElem;1;Size of array(◇apValores);1;-1)
   If(Is table number valid($vlElem))
      ◇apValores{$vlElem}:=Table($vlElem)
   End if
End for
```

ARRAY REAL

ARRAY REAL (nombreArray ; tamaño {; tamaño2})

Parámetro	Tipo		Descripción
nombreArray	Array	➡	Nombre del array
tamaño	Entero largo	➡	Número de elementos en el array o Número de filas si se especifica tamaño2
tamaño2	Entero largo	➡	Número de columnas en un array de dos dimensiones

Descripción

El comando **ARRAY REAL** crea y/o redimensiona un array de elementos de tipo **Real** en memoria.

- El parámetro nombreArray es el nombre del array.
- El parámetro *tamaño* es el número de elementos del array.
- El parámetro *tamaño2* es opcional; si se especifica *tamaño2*, el comando crea un array bidimensional. En este caso, *tamaño* especifica el número de filas y *tamaño2 e*specifica el número de columnas en cada array. Cada línea en un array de dos dimensiones puede tratarse como un elemento y un array. Esto significa que mientras trabaja con la primera dimensión del array, puede utilizar otros comandos de array para insertar y borrar arrays completos en un array de dos dimensiones.

Cuando aplica **ARRAY REAL** a un array existente:

- Si agranda el tamaño del array, los elementos existentes no son modificados y los nuevos elementos se inicializan en 0.
- Si reduce el tamaño del array, los últimos elementos son borrados del array y se pierden.

Ejemplo 1

Este ejemplo crea un array de proceso de 100 elementos de tipo Real:

```
ARRAY REAL(arValores;100)
```

Ejemplo 2

Este ejemplo crea un array local de 100 filas de 50 elementos de tipo Real:

```
ARRAY REAL($arValores;100;50)
```

Ejemplo 3

Este ejemplo crea un array interproceso de 50 elementos de tipo Real y asigna a cada elemento su número:

```
ARRAY REAL(◊arValores;50)
For($vlElem;1;50)
   ◊arValores{$vlElem}:=$vlElem
End for
```

ARRAY TEXT

ARRAY TEXT (nombreArray ; tamaño {; tamaño2})

Parámetro	Tipo		Descripción
nombreArray	Array	➡	Nombre del array
tamaño	Entero largo	➡	Número de elementos en el array o Número de filas si se especifica tamaño2
tamaño2	Entero largo	➡	Número de columnas en un array bidimensional

Descripción

El comando **ARRAY TEXT** crea y/o redimensiona un array de elementos de tipo *Texto* en memoria.

- El parámetro *nombreArray* es el nombre del array.
- El parámetro *tamaño* es el número de elementos en el array.
- El parámetro *tamaño2* es opcional; si se especifica *tamaño2*, el comando crea un array bidimensional. En este caso, *tamaño* especifica el número de filas y *tamaño2* especifica el número de columnas en cada array. Cada fila en un array bidimensional puede tratarse como un elemento y como un array. Esto significa que mientras trabaja con la primera dimensión del array, puede utilizar otros comandos de array para insertar y borrar arrays enteros en un array bidimensional.

Cuando aplica **ARRAY TEXT** a un array existente:

- Si agranda el tamaño del array, los elementos existentes no se modifican, y los nuevos elementos son inicializados en "" (cadena vacía).
- Si reduce el tamaño del array, se pierden los últimos elementos borrados del array.

Ejemplo 1

Este ejemplo crea un array de proceso de 100 elementos de tipo *Texto*:

```
ARRAY TEXT(atValores;100)
```

Ejemplo 2

Este ejemplo crea un array local de 100 filas de 50 elementos de tipo *Texto*:

```
ARRAY TEXT($atValores;100;50)
```

Ejemplo 3

Este ejemplo crea un array interproceso de 50 elementos de tipo texto y asigna a cada elemento el valor "Elemento #" seguido por su número de elemento:

```
ARRAY TEXT(◊atValues;50)
For($vlElem;1;50)
   ◊atValores{$vlElem}:="Elemento #"+String($vlElem)
End for
```

ARRAY TIME

ARRAY TIME (nomArray ; tam {; tam2})

Parámetro	Tipo		Descripción
nomArray	Array	➡	Nombre de array
tam	Entero largo	➡	Número de elementos en el array o Número de filas si se especifica tamaño2
tam2	Entero largo	➡	Número de columnas en un array de dos dimensiones

Descripción

El comando **ARRAY TIME** crea o redimensiona una array de tipo tiempo en memoria.
Recordatorio: en 4D, las horas pueden ser procesados como valores numéricos . En las versiones de 4D anteriores a v14, había que combinar un array entero largo con un formato de visualización para gestionar una array de horas.

El parámetro *nomArray* es el nombre del array.

El parámetro *tam* es el número de elementos del array.

El parámetro *tam2* es opcional. Si lo pasa, este comando crea un array de dos dimensiones. En este caso, *tam* especifica el número de filas y *tam2* el número de columnas de cada array. Cada fila en un array de dos dimensiones se puede procesar tanto como un elemento y como un array. Esto significa que cuando se trabaja con la primera dimensión de un array de dos dimensiones, puede insertar y retirar arrays enteros utilizando otros comandos de este tema .

Cuando aplica el comando **ARRAY TIME** a un array existente :

- Si amplía su tamaño, los elementos existentes no cambian y los nuevos elementos se inicializan en el valor de hora nulo (00:00:00) .
- Si reduce su tamaño, se eliminan y pierden los elementos de abajo del array.

Cuando aplica **SELECTION TO ARRAY** o **SELECTION RANGE TO ARRAY** a un campo de tipo Hora, tenga en cuenta que sólo crean un array de tipo Hora si el array no se ha definido como otro tipo, tal como Entero largo, por ejemplo.

Ejemplo 1

Este ejemplo crea un array proceso que contiene 100 elementos de tipo Hora:

```
ARRAY TIME(arrTimes;100)
```

Ejemplo 2

Este ejemplo crea un array local de 100 filas, conteniendo cada una 50 elementos de tipo Hora:

```
ARRAY TIME($arrTimes;100;50)
```

Ejemplo 3

Como los arrays de horas aceptan valores numéricos, el siguiente código es válido:

```
ARRAY TIME($arrTimeValues;10)
$CurTime:=Current time+1
APPEND TO ARRAY($arrTimeValues;$CurTime)
$Found:=Find in array($arrTimeValues;$CurTime)
```

ARRAY TO LIST

ARRAY TO LIST (array ; lista {; refElementos})

Parámetro	Tipo		Descripción
array	Array	➡	Array del cual copiar los elementos del array
lista	Cadena, ListRef	➡	Lista en la cual copiar los elementos del array
refElementos	Array	➡	Array numérico de números de referencia de los elementos

Descripción

El comando **ARRAY TO LIST** crea o reemplaza la lista jerárquica o la *lista* utilizando los elementos del *array.*

Puede pasar en el parámetro opcional *lista*, una lista de selección (cadena) o una referencia de lista jerárquica (*refLista*). En el segundo caso, esta lista debe haber sido creada previamente (por ejemplo utilizando el comando **New list**) para que este comando funcione.

El parámetro opcional *refElementos*, si se pasa, debe ser un array numérico sincronizado con el array *array*. Cada elemento de este array indica el número de referencia del elemento de la lista correspondiente en *array*. Si omite este parámetro, 4D define automáticamente los números de referencia de los elementos de la lista 1, 2.

Nota de compatibilidad: el comando **ARRAY TO LIST** debe utilizarse con precaución por las siguientes limitaciones:

- Este comando permite definir solamente los elementos del primer nivel de la lista.
- cuando lo utiliza con una lista, este comando modifica la estructura de la aplicación (las listas se guardan en el archivo de estructura), las modificaciones efectuadas en local se pierden durante la actualización del archivo de estructura en producción.
- Este comando no puede utilizarse en un componente con una lista de selección ya que los componentes se cargan con la estructura en sólo lectura.

Puede utilizar **ARRAY TO LIST** para construir una lista basada en los elementos de un array. Sin embargo, para librarse de estas restricciones y explotar por completo las listas de valores, le recomendamos utilizar los comandos del tema **Listas jerárquicas**.

Ejemplo

El siguiente ejemplo copia el array *atRegiones* en la lista "Regiones:"

```
ARRAY TO LIST(atRegions;"Regions")
```

Ejemplo

Usted quiere poner los distintos valores de un campo en una lista, por ejemplo para crear un menú pop-up jerárquico. Puede escribir:

```
ALL RECORDS([Empresa])
DISTINCT VALUES([Empresa]pais;$arrPaises)
ListaPais:=New list
ARRAY TO LIST($arrPaises;ListaPais)
```

Gestión de errores

El comando **ARRAY TO LIST** genera el error *-9957* cuando se aplica a una lista que está siendo editada en el editor de listas del entorno Diseño. Puede interceptar este error utilizando un método de proyecto **ON ERR CALL**.

ARRAY TO SELECTION

ARRAY TO SELECTION {(array ; campo {; array2 ; campo2 ; ... ; arrayN ; campoN}{; *})}

Parámetro	Tipo		Descripción
array	Array	➡	Array a copiar en la selección
campo	Campo	⬅	Campo a recibir los valores del array
*	Operador	➡	Esperar ejecución

Descripción

El comando **ARRAY TO SELECTION** copia uno o más arrays en una selección de registros. Todos los campos listados deben pertenecer a la misma tabla.

Si una selección existe en el momento del llamado, los elementos del array se colocan en los registros, basados en el orden del array y en el orden de los registros. Si hay más elementos que registros, se crean nuevos registros. Los registros, bien sean nuevos o existentes, se guardan automáticamente.

Nota: dado que puede crear nuevos registros, este comando no tiene estado de sólo lectura de la tabla (si la hay) (ver **Record Locking**).

Todos los arrays deben tener el mismo número de elementos. Si los arrays son de diferentes tamaños, se genera un error de sintaxis

Este comando efectúa la operación inversa de **SELECTION TO ARRAY**. Sin embargo, el comando **ARRAY TO SELECTION** no permite utilizar los campos de diferentes tablas, incluyendo tablas relacionadas, incluso cuando existe una relación automática.

Si pasa el parámetro *, 4D no ejecuta inmediatamente la línea de instrucción correspondiente pero la guarda en memoria; de esta forma puede apilar varias líneas que terminen en *. El conjunto de las líneas en espera es ejecutado por una instrucción **ARRAY TO SELECTION** final sin parámetro *. Por esta razón, ahora el comando puede llamarse sin parámetros.
Como para el comando **QUERY**, este principio permite romper una instrucción compleja en un conjunto de líneas, lo cual es más fácil de leer y mantener. También es posible insertar instrucciones intermediarias.

Advertencia: utilice **ARRAY TO SELECTION** con precaución, porque este comando reemplaza la información de los registros existentes. Si un registro está bloqueado por otro proceso durante la ejecución de **ARRAY TO SELECTION**, ese registro no se modifica. Todos los registros bloqueados se colocan en LockedSet. Después de la ejecución de **ARRAY TO SELECTION**, puede probar el conjunto LockedSet para ver si contiene registros bloqueados.

Nota: este comando no tiene en cuenta el estado de sólo lectura/lectura-escritura de la tabla que contiene el campo.

4D Server: el comando es optimizado por 4D Server. Los arrays son enviados por el equipo cliente al servidor, y los registros son modificados o creados en el equipo servidor. Como tal solicitud es manejada de modo sincrónico, el equipo cliente debe esperar a que la operación se complete con éxito. En los entornos multiusuario o multiproceso, ningún registro bloqueado será sobrescrito.

Ejemplo 1

En el siguiente ejemplo, los arrays *asApellidos* y *asEmpresas* escriben datos en la tabla *[Personas]*. Los valores del array *asApellidos* se ubican en el campo *[Personas]Apellido* y los valores del array *asEmpresas* se ubican en el campo *[Personas]Empresa*:

```
ARRAY TO SELECTION(asApellidos;[Personas]Apellido;asEmpresas;[Personas]Empresa)
```

Ejemplo 2

Usted desea copiar una selección de registros a una tabla archivo seleccionando los campos de acuerdo con el valor de opción:

```
ARRAY TEXT($_name;0)
ARRAY TEXT($_firstname;0)
ARRAY TEXT($_cv;0)
ARRAY PICTURE($_photo;0)

SELECTION TO ARRAY([Candidate]Name;$_name;*)
SELECTION TO ARRAY([Candidate]Firstname;$_firstname;*)
If(withCV)  //load the CV field
   SELECTION TO ARRAY([Candidate]cv;$_cv;*)
End if
If(withPhoto)  //cargar campo de foto
   SELECTION TO ARRAY([Candidate]photo;$_photo;*)
End if
SELECTION TO ARRAY  //ejecutar copia

REDUCE SELECTION([Candidate_Archive];0)
ARRAY TO SELECTION($_name;[Candidate_Archive]Name;*)
ARRAY TO SELECTION($_prenom;[Candidate_Archive]Firstname;*)
If(withCV)
```

```
    ARRAY TO SELECTION($_cv;[Candidate_Archive]cv;*)
End if
If(withPhoto)
    ARRAY TO SELECTION($_photo;[Candidate_Archive]photo;*)
End if
ARRAY TO SELECTION
```

BOOLEAN ARRAY FROM SET

BOOLEAN ARRAY FROM SET (arrBool {; conjunto})

Parámetro	Tipo		Descripción
arrBool	Array booleano	⇐	Array para indicar si un registro está en un conjunto o no
conjunto	Cadena	⇒	Nombre del conjunto o UserSet si se omite este parámetro

Descripción

El comando **BOOLEAN ARRAY FROM SET** llena un array de booleanos indicando si cada registro en la tabla está o no en *conjunto*.

Los elementos en el array se ordenan en función del orden de creación de los registros en la tabla (números de registro absolutos). Si N es el número de registros en la tabla, el elemento 0 del array corresponde al registro número 0, el elemento 1 del array corresponde al registro número 1, etc.

Cada elemento del array es:

- **Verdadero** si el registro correspondiente pertenece al conjunto.
- **Falso** si el registro correspondiente no pertenece al conjunto.

Advertencia: el número total de elementos en el array *arrBool* no es significativo. Por razones estructurales, este número puede ser diferente del número de registros realmente presentes en la tabla. Los posibles elementos extras son definidos como **False**.

Si no pasa el parámetro *conjunto*, el comando utilizará **UserSet** en el proceso actual.

COPY ARRAY

COPY ARRAY (fuente ; destino)

Parámetro	Tipo		Descripción
fuente	Array	➡	Array a copiar
destino	Array	⬅	Array de destino

Descripción

El comando **COPY ARRAY** crea o remplaza el array *destino* con el mismo contenido, tamaño y tipo del array *fuente*.

Los arrays *fuente* y *destino* pueden ser locales, proceso o interproceso. El alcance del array no tiene importancia en el momento de copiar arrays.

Notas:

- En modo compilado, el array *destino* debe ser del mismo tipo que el array *fuente*.
- Al copiar arrays de objetos, sólo se duplican las referencias de los objetos que las contienen y no los objetos. Esto significa que cualquier modificación realizada en un objeto de un array se aplicará a todas las instancias existentes del objeto en los arrays copiados. Si necesita duplicar objetos, debe utilizar el comando **OB Copy**.

Nota: en modo compilado, el array *destino* debe ser del mismo tipo que el array *fuente*.

Ejemplo

El siguiente ejemplo llena el array C. Luego crea un nuevo array, llamado D, del mismo tamaño de C y con el mismo contenido:

```
ALL RECORDS([Personas]) ` Seleccionar todos los registros en Personas
SELECTION TO ARRAY([Personas]Empresa;C) ` Mover los datos del campo empresa al array C
COPY ARRAY(C;D) ` Copiar el array C al array D
```

Count in array

Count in array (array ; valor) -> Resultado

Parámetro	Tipo		Descripción
array	Array	➡	Array donde efectuar el conteo
valor	Expresión	➡	Valor a contar
Resultado	Entero largo	↩	Número de ocurrencias encontradas

Descripción

El comando **Count in array** devuelve el número de veces ocurrencias del *valor* en el *array*.

Este comando puede utilizarse con los siguientes tipos de array: Texto, Alfa, Numérico, Fecha, Puntero y Booleano. Los parámetros a*rray* y *valor* deben ser del mismo tipo o de un tipo compatible.

Si ningún elemento del *array* corresponde al *valor*, el comando devuelve 0.

Ejemplo

El siguiente ejemplo permite visualizar el número de líneas seleccionadas en un list box:

```
`tBList es el nombre de un array de una columna de un List box
ALERT(String(Count in array(tBList;True))+" línea(s) seleccionada(s) en el list box")
```

DELETE FROM ARRAY

DELETE FROM ARRAY (array ; posicion {; reemplazos})

Parámetro	Tipo		Descripción
array	Array	➡	Array del cual borrar elementos
posicion	Entero largo	➡	Elemento donde comienza la supresión
reemplazos	Entero largo	➡	Número de elementos a borrar, o 1 elemento si se omite

Descripción

El comando **DELETE FROM ARRAY** borra uno o más elementos del *array*. Los elementos se borran a partir del elemento especificado por *donde*.

El parámetro *cuantos* es el número de elementos a borrar. Si no se especifica *cuantos*, entonces se borra un elemento. El tamaño del array se reduce en *cuantos*.

Ejemplo 1

El siguiente ejemplo borra tres elementos, comenzando en el elemento 5:

```
DELETE FROM ARRAY(anArray;5;3)
```

Ejemplo 2

El siguiente ejemplo borra el último elemento de un array, si existe:

```
$vlElem:=Size of array(anArray)
If($vlElem>0)
    DELETE FROM ARRAY(anArray;$vlElem)
End if
```

DISTINCT ATTRIBUTE PATHS

DISTINCT ATTRIBUTE PATHS (campoObjeto ; arrayRuta)

Parámetro	Tipo		Descripción
campoObjeto	Campo	➡	Campo objeto indexado
arrayRuta	Array texto	⬅	Array para recibir la lista de rutas diferentes

Descripción

El comando **DISTINCT ATTRIBUTE PATHS** devuelve la lista de rutas distintas que se encuentran en el campo objeto indexado pasado en *campoObjeto* para la selección actual de la tabla a la que pertenece el campo.

En *campoObjeto*, debe pasar un campo de tipo Objeto indexado; de lo contrario se devuelve un error.

Después de la llamada, el tamaño de *arrayRuta* es igual al número de rutas distintas que se encuentran en la selección. Las rutas a atributos de objetos anidados se devuelven utilizando la notación estándar punto, por ejemplo "empresa.direccion.numero". Tenga en cuenta que los nombres de atributo de objeto son sensibles a las mayúsculas y minúsculas. El comando no cambia la selección actual o el registro actual.

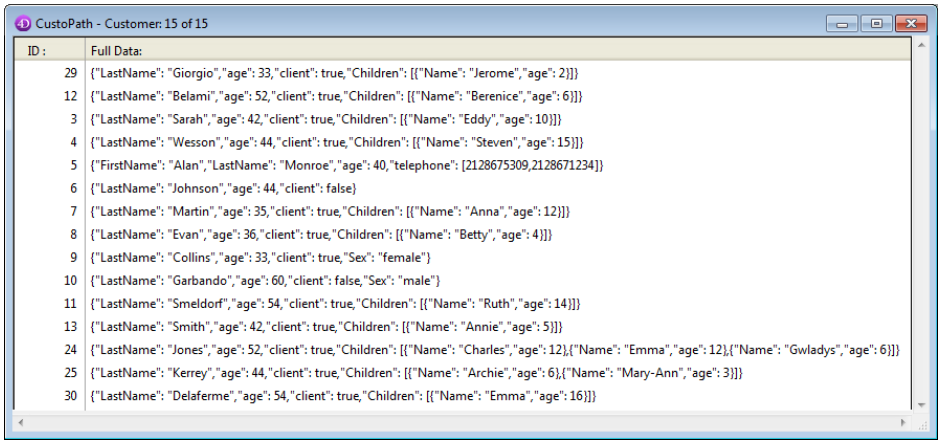
En *arrayRuta*, se devuelve la lista de rutas distintas en orden alfabético (diacrítico).

Notas:

- los registros con un valor indefinido en campoObjeto no se tienen en cuenta.
- Las rutas de atributos creadas durante una transacción son tenidas en cuenta por el comando. Es importante señalar que estas rutas se conservan en el índice del campo objeto, incluso si la transacción ha sido cancelada.

Ejemplo

Su base contiene un campo objeto [Customer]full_Data (indexado) con 15 registros:



Si ejecuta este código:

```
ARRAY TEXT(aTPaths;0)
ALL RECORDS([Customer])
DISTINCT ATTRIBUTE PATHS([Customer]full_Data;aTPaths)
```

El array *aTPaths* obtiene los siguientes elementos:

Elemento	Valor
1	"age"
2	"Children"
3	"Children[]"
4	"Children[].age"
5	"Children[].Name"
6	"Children.length"
7	"client"
8	"FirstName"
9	"LastName"
10	"Sex"
11	"telephone"
12	"telephone[]"
13	"telephone.length"

Nota: "length" es una *propiedad virtual* que está disponible automáticamente para todos los atributos de tipo array. Ofrece el tamaño del array, es decir, el número de elementos,y puede ser utilizado en las peticiones. Para mayor información, consulte el párrafo **Utilización de la propiedad virtual .length**.

DISTINCT ATTRIBUTE VALUES

DISTINCT ATTRIBUTE VALUES (campoObjeto ; ruta ; arrayValores)

Parámetro	Tipo		Descripción
campoObjeto	Campo	➡	Campo de objeto del que desea obtener la lista de valores de atributos distintos
ruta	Texto	➡	Ruta de acceso del atributo cuyos valores distintos desea obtener
arrayValores	Array texto, Array entero largo, Array booleano, Array fecha, Time array	⬅	Valores distintos en la ruta de atributos

Descripción

Tema: Arrays
El comando **DISTINCT ATTRIBUTE VALUES** crea y llena el *arrayValores* con valores no repetidos (únicos) procedentes del atributo *ruta* en el campo *campoObjeto* para la selección actual de la tabla a la que pertenece el campo. Si *campoObjeto* no es un campo de objeto indexado, se devuelve un error.

Pase en *ruta* una ruta atributo válida. Utilice la notación punto estándar para definir rutas a atributos anidados, por ejemplo "empresa.direccion.numero". Tenga en cuenta que los nombres de atributos objeto son sensibles a las mayúsculas y minúsculas.

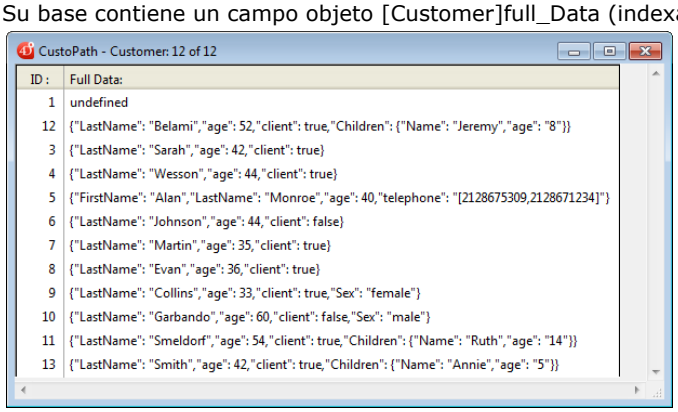
El array que pasó en *arrayValores* debe ser del mismo tipo que el atributo *ruta* pasado como parámetro, de lo contrario el array se vuelve a digitar. Sólo se admiten los siguientes tipos de arrays: numérico, texto, fecha y hora (los valores deben ser escalar: punteros, objetos, blobs o imágenes no son soportados).
Después de la llamada, el tamaño del array es igual al número de valores distintos que se encuentran en la selección. El comando no cambia la selección actual o el registro actual.

Utilización de la propiedad virtual .length

Puede utilizar la propiedad virtual "longitud" con este comando. Está disponible automáticamente para todos los atributos de tipo array, y ofrece el tamaño del array, es decir, el número de elementos que contiene. Esta propiedad está diseñada para ser utilizada en búsquedas (ver **QUERY BY ATTRIBUTE**QUERY BY ATTRIBUTE). También se puede utilizar con el comando **DISTINCT ATTRIBUTE VALUES** para obtener los diferentes tamaños de array para un atributo.

Ejemplo

Su base contiene un campo objeto [Customer]full_Data (indexado) con 12 registros:



Si ejecuta este código:

```
ARRAY LONGINT(aLAges;0)
ALL RECORDS([Customer])
 //get the distinct values for the "age" attibute
DISTINCT ATTRIBUTE VALUES([Customer]full_Data;"age";aLAges)
```

El array *aLAges* obtiene los siguientes 9 elementos:

```
//aLAges{1}=33
//aLAges{2}=35
//aLAges{3}=36
//aLAges{4}=40
//aLAges{5}=42
//aLAges{6}=44
//aLAges{7}=52
//aLAges{8}=54
//aLAges{9}=60
```

DISTINCT VALUES

DISTINCT VALUES (unCampo ; array {; contArray})

Parámetro	Tipo		Descripción
unCampo	Campo	➡	Campo o subcampo indexable a utilizar para datos
array	Array	⬅	Array a recibir los datos del campo
contArray	Array entero largo, Array real	⬅	Array a recibir el número de ocurrencias de cada valor

Descripción

El comando **DISTINCT VALUES** crea y llena el array *array* con todos valores distintos (únicos) del campo *campo* para la selección actual de la tabla del campo y, opcionalmente, devuelve en *contArray* el número de ocurrencias de cada valor.

Puede pasar a **DISTINCT VALUES** todo tipo de campo o subcampo **indexable**, es decir, cuyo tipo soporte indexación sin que esté indexado necesariamente.
Sin embargo, la ejecución de este comando con campos no indexados será más lenta. También observe que en este caso, el comando pierde el registro actual.
DISTINCT VALUES analiza y extrae los valores no repetidos en los registros seleccionados únicamente.

Nota: cuando el comando **DISTINCT VALUES** se llama durante una transacción (que no ha terminado aún), el comando **tiene en cuenta** los registros creados durante esa transacción.

El array utilizado por **DISTINCT VALUES** debe ser del mismo tipo que el campo pasado como primer parámetro, de lo contrario el array se vuelve a digitar. Hay una excepción a esta regla:si el campo es de tipo Imagen (y está asociado a un índice de palabras claves), el array correspondiente debe ser de tipo Texto.

Después del llamado, el tamaño del array es igual al número de valores distintos encontrados en la selección. El comando no cambia la selección actual ni el registro actual. El comando **DISTINCT VALUES** utiliza el índice del campo, de manera que los elementos en *array* se devuelven ordenados en orden ascendente. Si este es el orden que usted necesita, no es necesario llamar a **SORT ARRAY** después de utilizar **DISTINCT VALUES**.

Nota: cuando se ejecuta **DISTINCT VALUES** se ejecuta con un campo de texto o imagen asociado a un índice de palabras claves, el comando llena el array con las palabras claves del índice. A diferencia de otros tipos de datos, los valores devueltos difieren de acuerdo a la existencia del índice. El índice de palabras claves siempre se tiene en cuenta, incluso cuando el campo está asociado a un índice estándar. Si el campo texto o imagen no está asociado a un índice de palabras claves, el array se devuelve vacío.

El comando acepta un array *contArray* como un parámetro opcional. Cuando se pasa, este array devuelve, para cada valor no repetido en *campo*, el número de ocurrencias detectadas en la selección actual. El array *contArray* Se dimensiona automáticamente al número de elementos en *array*. Por ejemplo, para una selección que contiene tres registros con los valores de campos "A", "B" y "A", *array* contendrá {A;B} y *contArray* contendrá {2;1}. Puede pasar un array Entero largo o Real en *contArray*.

Nota: el parámetro *contArray* no es soportado para los campos texto o imagen asociados a los índices de palabras claves (en este contexto, se devuelve vacío).
Advertencia: DISTINCT VALUES puede crear array grandes, dependiendo del tamaño de la selección y del número de valores diferentes en los registros. Los arrays residen en memoria, por lo tanto es buena idea probar el resultado después de la ejecución del comando. Para hacer esto, pruebe el tamaño del array resultante o cubra la llamada al comando, utilizando un método de proyecto **ON ERR CALL**.

4D Server: este comando es optimizado por 4D Server. El array se crea y los valores son calculados en el servidor; luego el array se envía, en su totalidad, al cliente.

Nota: este comando no soporta campos de tipo Objeto.

Ejemplo 1

El siguiente ejemplo crea una lista de ciudades a partir de la selección actual e indica al usuario el número de ciudades en las cuales la empresa tiene almacenes:

```
ALL RECORDS([Almacenes]) ` Crear una selección de registros
DISTINCT VALUES([Almacenes]Ciudad;asCiudades)
ALERT("La empresa tiene almacenes en "+String(Size of array(asCiudades))+" ciudades.")
```

Ejemplo 2

Usted quiere obtener la lista completa de palabras claves contenidas en el índice de palabras claves del campo "Fotos":

```
ALL RECORDS([IMAGES])
ARRAY TEXT(<>_MyKeywords;10)
DISTINCT VALUES([IMAGES]Fotos;<>_MyKeywords)
```

Ejemplo 3

Para calcular las estadísticas, usted desea ordenar el número de valores distintos en un campo en orden descendente:

```
ARRAY TEXT($_issue_type;0)
ARRAY LONGINT($_issue_type_instance;0)
DISTINCT VALUES([Issue]iType;$_issue_type;$_issue_type_instances)
SORT ARRAY($_issue_type_instances;$_issue_type;<)
```

Find in array

Find in array (array ; valor {; inicio}) -> Resultado

Parámetro	Tipo		Descripción
array	Array	→	Array a buscar
valor	Expresión	→	Valor del mismo tipo a buscar en el array
inicio	Entero largo	→	Elemento a partir del cual comenzar la búsqueda
Resultado	Entero largo	↩	Número del primer elemento en el array que corresponde al valor

Descripción

El comando **Find in array** devuelve el número del primer elemento del *Array* que corresponde a *valor*.

Find in array puede utilizarse con arrays de tipo Texto, Alfa, Numérico, Fecha, Puntero, y Booleano. Los parámetros *Array* y *valor* deben ser del mismo tipo.

valor debe coincidir exactamente con el elemento a encontrar (se aplican las mismas reglas que para el operador de igualdad, ver **Operadores de comparación**). Si no se encuentra ningún elemento, **Find in array** devuelve –1.

Si se especifica *principio*, el comando comienza la búsqueda en el número de elemento especificado por *principio*. Si no se especifica *principio*, el comando comienza la búsqueda en el elemento 1.

Ejemplo 1

El siguiente método de proyecto borra todos los elementos vacíos del array alfa o texto cuyo puntero se pasa como parámetro:

```
  ` Método de proyecto LIMPIAR ARRAY
  ` LIMPIAR ARRAY ( Puntero )
  ` LIMPIAR ARRAY ( -> Array Texto o Alfa )

C_POINTER($1)
Repeat
   $vlElem:=Find in array($1->;"")
   If($vlElem>0)
      DELETE FROM ARRAY($1->;$vlElem)
   End if
Until($vlElem<0)
```

Después de implementar este método de proyecto en una base, puede escribir:

```
ARRAY TEXT(atValores;...)
  ` ...
  ` Utilizar el array como quiera
  ` ...
  ` Eliminar los elementos de cadenas vacías
LIMPIAR ARRAY(->atValores)
```

Ejemplo 2

El método de proyecto siguiente selecciona el primer elemento de un array cuyo puntero pasado como primer parámetro corresponde al valor de la variable o del campo cuyo puntero se pasa como parámetro:

```
  ` Método de proyecto SELECCIONAR ELEMENTO
  ` SELECCIONAR ELEMENTO( Puntero ; Puntero)
  `SELECCIONAR ELEMENTO ( -> Array Texto o Alfa ; -> Campo o variable de tipo Texto o Alfa )

$1->:=Find in array($1->;$2->)
If($1->=-1)
   $1->:=0  ` Si no se encuentra un elemento, fijar el array en un elemento no seleccionado
End if
```

Después de implementar este método proyecto en la base, puede escribir por ejemplo:

```
  ` Método de objeto de menú desplegable asGenero
Case of
   :(Form event=On Load)
      SELECCIONAR ELEMENTO(->asGenero;->[Personas]Genero)

End case
```

Nota: este ejemplo utiliza el **elemento seleccionado** del array. Tenga en cuenta que el elemento seleccionado no es significativo si el array contiene más de 32.767 elementos (ver **Arrays y objetos de formulario**). En este caso, es necesario utilizar una variable de tipo entero largo para almacenar el resultado de **Find in array**.

Find in sorted array

Find in sorted array (array ; valor ; > or < {; posPrim {; posUlt}}) -> Resultado

Parámetro	Tipo		Descripción
array	Array	➡	Array a buscar
valor	Expresión	➡	Valor del mismo tipo a buscar en el array
> or <	Operador	➡	> si el array está en orden ascendente, < si el está en orden descendente
posPrim	Entero largo	⬅	Posición de su primera aparición si se encuentra el valor; de lo contrario la posición donde debe insertarse el valor
posUlt	Entero largo	⬅	Posición de su última ocurrencia si se encuentra el valor; de lo contrario lo mismo que posPrim
Resultado	Booleano	↩	True si al menos un elemento en el array corresponde al valor, de lo contrario False

Descripción

Tema: Arrays

El nuevo comando **Find in sorted array** devuelve **true** si al menos un elemento en el *array* ordenado con el *valor* y opcionalmente devuelve la posición de los elementos correspondientes. A diferencia de **Find in array**, **Find in sorted array** sólo funciona con un *array* ordenado y ofrece información sobre la posición de las ocurrencias, lo que le permite insertar elementos si es necesario.

El *array* debe estar ya ordenado y debe coincidir con el orden especificado por el parámetro *> or <* (es decir, el símbolo "mayor que" para el orden ascendente y el símbolo "menor que" para el orden descendente). El comando **Find in sorted array** tomará ventaja de la ordenación y uso de un algoritmo de *búsqueda binaria*, que es mucho más eficiente para grandes arrays (para más información, consulte la página algoritmo de búsqueda binaria en Wikipedia). Sin embargo, si el array no está ordenado correctamente, el resultado puede ser incorrecto.

El comando ignorará la indicación de ordenación y se comporta como un **Find in array** estándar (búsqueda secuencial, devolviendo -1 para *posPrim* y *posUlt* si no se encuentra el *valor*) en cualquiera de los siguientes casos:

- si el tipo de array no se puede ordenar (por ejemplo arrays puntero),
- si el array es de tipo booleano (no preciso),
- si la base de datos no es Unicode (modo compatibilidad) y el array es una cadena o array texto,
- cuando se busca en un array text para una cadena que incluye un comodín ('@') al principio o en el medio de la cadena (utilizando una búsqueda binaria con un carácter comodín no es posible porque los elementos correspondientes pueden no ser contiguos en el array).

En caso de que el comando devuelva **False**, el valor devuelto en *posPrim* se puede pasar a **INSERT IN ARRAY** para insertar el *valor*en el array manteniendo el array ordenado. Esta secuencia es más rápida que la inserción de un nuevo elemento al final del array y luego llamar a **SORT ARRAY** para moverlo al lugar correcto.

El valor devuelto en *posUlt* se puede utilizar en conjunto con el valor devuelto en *posPrim* para iterar sobre cada elemento del array correspondiente al *valor* (vía un **For...End for**) o para encontrar el número total de ocurrencias (como lo encontraría **Count in array**, pero más rápido).

Ejemplo 1

Usted desea insertar un valor, si es necesario, manteniendo el array ordenado:

```
C_LONGINT($pos)
If(Find in sorted array($array ;$value ;>;$pos)
    ALERT("Found at pos "+String($pos))
Else
    INSERT IN ARRAY($array ;$pos)
    $array{$pos}:=$value
End if
```

Ejemplo 2

Usted quiere encontrar el número de ocurrencias de las cadenas que comienzan por "test" y crear una cadena que concatena todos estos elementos:

```
C_LONGINT($posFirst ;$posLast)
C_TEXT($output)
If(Find in sorted array($array ;"test@";>;$posFirst ;$posLast))
    $output:="Found "+String($posLast-$posFirst+1)+" results :\n"
End if
For($i ;$posFirst ;$posLast)
    $output:=$output+$array{$i}+"\n"
End for
```

INSERT IN ARRAY

INSERT IN ARRAY (array ; posicion {; reemplazos})

Parámetro	Tipo		Descripción
array	Array	➡	Nombre del array
posicion	Entero largo	➡	Donde insertar los elementos
reemplazos	Entero largo	➡	Número de elementos a insertar, o 1 elemento si se omite

Descripción

El comando **INSERT IN ARRAY** inserta uno o más elementos en el *array*. Los nuevos elementos se insertan antes del elemento especificado por *donde*, y se inicializan en el valor vacío del tipo de array. Todos los elementos más allá de *donde* se mueven consecuentemente en el array por un valor de uno o por el valor especificado en *cuantos*.

Si *donde* es mayor que el tamaño del array, los elementos se añaden al final del array.

El parámetro *cuantos* es el número de elementos a insertar. Si no se especifica *cuantos*, entonces se inserta sólo un elemento. El tamaño del array aumenta en *cuantos*.

Ejemplo 1

El siguiente ejemplo inserta cinco nuevos elementos, comenzando en el elemento 10:

```
INSERT IN ARRAY(anArray;10;5)
```

Ejemplo 2

El siguiente ejemplo añade un elemento a un array:

```
$vlElem:=Size of array(anArray)+1
INSERT IN ARRAY(anArray;$vlElem)
anArray{$vlElem}:=...
```

LIST TO ARRAY

LIST TO ARRAY (lista ; array {; refElementos})

Parámetro	Tipo		Descripción
lista	Cadena, ListRef	➡	Lista de la cual copiar los elementos de primer nivel
array	Array	⬅	Array al cual copiar los elementos de la lista
refElementos	Array	⬅	Números de referencia de los elementos de la lista

Descripción

El comando **LIST TO ARRAY** crea o reemplaza el array a*rray* con los elementos del primer nivel de la lista o de la lista de selección designada por *lista*.

En el parámetro *lista* puede pasar el nombre de una lista de selección (cadena), o una referencia de lista jerárquica (*RefList*).

Si no define previamente el array como de tipo Alfa o Texto, **LIST TO ARRAY** crea un nuevo array de tipo Texto por defecto.

Nota: en modo compilado, el *array* debe haber sido definido previamente y no puede ser digitado nuevamente.

El parámetro opcional *refElements* (un array de tipo numérico) devuelve los números de referencia de los elementos de la lista.

Puede utilizar **LIST TO ARRAY** para construir un array basado en los elementos de primer nivel de una lista. Sin embargo, este comando no le permite trabajar con los elementos de las sublistas. Para trabajar con listas jerárquicas, utilice los comandos de listas jerárquicas, en particular **Load list**.

Ejemplo 1

El siguiente ejemplo copia los elementos de una lista llamada Regiones en el array llamado *atRegions*:

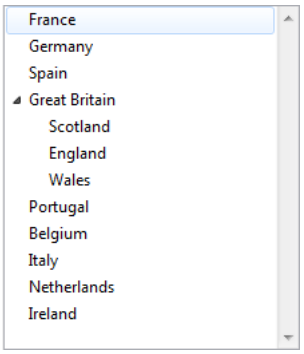
```
LIST TO ARRAY("Regions";atRegions)
```

Ejemplo 2

Dada una lista jerárquica creada como se ve a continuación:

```
myList2:=New list
APPEND TO LIST(myList2;"Scotland";1)
APPEND TO LIST(myList2;"England";2)
APPEND TO LIST(myList2;"Wales";3)
myList1:=New list
APPEND TO LIST(myList1;"France";1)
APPEND TO LIST(myList1;"Germany";2)
APPEND TO LIST(myList1;"Spain";3)
APPEND TO LIST(myList1;"Great Britain";4;myList2;True)
APPEND TO LIST(myList1;"Portugal";5)
APPEND TO LIST(myList1;"Belgium";6)
APPEND TO LIST(myList1;"Italy";7)
APPEND TO LIST(myList1;"Netherlands";8)
APPEND TO LIST(myList1;"Ireland";9)
```

Esta lista puede estar representada como:



Si ejecuta la siguiente instrucción:

```
LIST TO ARRAY(myList1;$MyArray)
```

...obtiene

```
$MyArray{1}="France"
$MyArray{2}="Germany"
$MyArray{3}="Spain"
$MyArray{4}="Great Britain"
$MyArray{5}="Portugal"
...
```

LONGINT ARRAY FROM SELECTION

LONGINT ARRAY FROM SELECTION (tabla ; arrReg {; seleccion})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la selección actual
arrReg	Array entero largo	⬅	Array de números de registros
seleccion	Cadena	➡	Nombre de la selección temporal o de la selección actual si este parámetro es omitido

Descripción

El comando **LONGINT ARRAY FROM SELECTION** llena el array *arrReg* con los números (absolutos) de los registros que están en *seleccion*.

Si no pasa el parámetro *seleccion*, el comando utiliza la selección actual de la *tabla*.

Nota: el elemento número 0 del array se inicializa en -1.

Ejemplo

Quiere recuperar los números de los registros de la selección actual:

```
ARRAY LONGINT($_arrRecNum;0)  //obligatorio para el modo compilado
LONGINT ARRAY FROM SELECTION([Clients];$_arrRecNum)
```

MULTI SORT ARRAY

MULTI SORT ARRAY (array {; orden}{; array2 ; orden2 ; ... ; arrayN ; ordenN})

Parámetro	Tipo		Descripción
array	Array	➡	Array(s) a ordenar
orden	Operador	➡	">" efectuar una ordenación creciente o "<" efectuar una ordenación decreciente Si se omite "aa" no ordenación

MULTI SORT ARRAY (ptrArrayNombre ; ordenArrayNombre)

Parámetro	Tipo		Descripción
ptrArrayNombre	Array puntero	➡	Array de punteros de arrays
ordenArrayNombre	Array entero largo	➡	Array de criterio de ordenación (1= ordenar por orden creciente, -1= ordenar por orden decreciente), 0= sincronización con ordenaciones anteriores

Descripción

El comando MULTI SORT ARRAY permite efectuar una ordenación multicriterios en un conjunto de arrays.

Este comando admite dos sintaxis diferentes.

- **Primera sintaxis: MULTI SORT ARRAY (Array{; orden}{; array2; sort2; ...; arrayN; ordenN})**

Esta sintaxis es la más sencilla; le permite pasar directamente los nombres de los arrays sincronizados a donde usted quiere aplicar una ordenación multicriterio.

Puede pasar un número ilimitado de parejas (*Array;> o <*) y/o sólo arrays. Todos los arrays pasados como parámetros se ordenan en de una manera sincronizada.

Puede pasar array de cualquier tipo excepto arrays tipo puntero o imagen. Puede ordenar un elemento de un array bidimensional (es decir *a2DArray{$vlEsteElemento}*), pero no puede ordenar el array bidimensional en si mismo (es decir *a2DArray*).

Para utilizar los contenidos de un array como criterio de ordenación, pase el parámetro *orden*. El valor del parámetro (*> o <*) determina el orden (creciente o decreciente) en el cual el array será ordenado. Si se omite el parámetro *orden*, el contenido del array no se utiliza como criterio de ordenación.

Nota: recuerde que debe pasarse al menos un criterio de ordenación para que el comando funcione. Si no se define un criterio de ordenación, se genera un error.

Los niveles de ordenación se determinan por el orden en el cual los arrays se pasan al comando: la posición de un array con criterio de ordenación en la sintaxis determina su nivel de ordenación.

- **Segunda sintaxis: MULTI SORT ARRAY (ptrArrayNombre; ordenArrayNombre)**

Esta sintaxis, más compleja, es muy valiosa para los desarrollos genéricos (por ejemplo, usted puede crear un método genérico para ordenar arrays de todo tipo, o una vez más crear el equivalente de un comando **SORT ARRAY** genérico).

El parámetro *ptrArrayNombre* contiene el nombre de un array de punteros de array; cada elemento de este array es un puntero designando un array a ordenar. Las ordenaciones serán efectuadas en el orden de los punteros de array definidos por *ptrArrayNombre*. **Advertencia:** todos los arrays apuntados por *ptrArrayNombre* deben tener el mismo número de elementos.

Nota: *ptrArrayNombre* puede ser un array de punteros local ($ptrArrayNombre), proceso (ptrArrayNombre) o interproceso (<>ptrArrayNombre). En cambio, los elementos de este array deben apuntar a arrays de proceso o interproceso únicamente.

El parámetro *ordenArrayNombre* contiene el nombre de un array en el cual cada elemento indica el criterio de ordenación (-1, 0 o 1) del elemento del array de punteros correspondiente:

-1 = Ordenación decreciente.
0 = El array no se utiliza como criterio de ordenación pero debe ser ordenado en función de otras ordenaciones.
1 = Ordenación creciente.

Nota: no es posible ordenar arrays de tipo Puntero o Imagen. Puede ordenar un elemento de un array de bidimensional (es decir a2DArray{$vlEsteElemento}), pero no puede ordenar el array bidimensional en sí mismo (es decir a2DArray).

Para cada elemento del array *ptrArrayNombre*, debe corresponder un elemento del array *ordenArrayNombre*. Por lo tanto los dos arrays deben tener exactamente el mismo número de elementos.

Ejemplo 1

El siguiente ejemplo utiliza la primera sintaxis: crea cuatro arrays y los ordena por ciudad (orden ascendente) luego por salario (orden descendente) con los dos últimos arrays *nombres_array* y *telNum_array*, siendo sincronizados de acuerdo con los criterios de ordenación anteriores:

```
ALL RECORDS([Empleados])
SELECTION TO ARRAY([Empleados]Ciudad;ciudades;[Empleados]Salario;salarios;[Empleados]Nombre;
nombres;[Empleados]TelNum;telNums)
MULTI SORT ARRAY(ciudades;>;salarios;<;nombres;telNums)
```

Si quiere que el array de nombres sea utilizado como tercer criterio de ordenación, sólo añada *> o <* después del parámetro *nombres_array.*
Observe que la sintaxis:

```
MULTI SORT ARRAY(ciudades;>;salarios;nombres;telNums)
```

es equivalente a:

```
SORT ARRAY(ciudades;salarios;nombres;telNums;>)
```

Ejemplo 2

El siguiente ejemplo utiliza la segunda sintaxis: crea cuatro arrays y los ordena por ciudad (orden creciente) y empresa (orden decreciente); los últimos dos arrays, nombres_Array y telNum_Array, están sincronizados de acuerdo al criterio de ordenación anterior:

```
ALL RECORDS([Empleados])
SELECTION TO ARRAY([Empleados]Ciudad;ciudades;[Empleados]Empresa;empresas;[Empleados]Nombre;
nombres;[Empleados]TelNum;telNums)
ARRAY POINTER(punteros_Array;4)
ARRAY LONGINT(sorts_Array;4)
Array_punteros{1}:=->ciudades
Array_orden{1}:=1
Array_punteros{2}:=->empresas
Array_orden{2}:=-1
Array_punteros{3}:=->nombres
Array_orden{3}:=0
Array_punteros{4}:=->telNums
Array_orden{4}:=0
MULTI SORT ARRAY(Array_punteros;Array_orden)
```

Si quiere que el array de nombres se utilice como tercer criterio de ordenación, debe asignar el valor 1 al elemento Array_orden{3}. O bien, si quiere que los arrays se ordenen sólo por el criterio ciudades, asigne el valor 0 a los elementos Array_orden{2}, Array_orden{3} y Array_orden{4}. De esta manera, puede obtener un resultado idéntico a **SORT ARRAY(ciudades;empresas;nombres;telNums;>)**.

SELECTION RANGE TO ARRAY

SELECTION RANGE TO ARRAY (inicio ; fin {; campo ; array} {; campo2 ; array2 ; ... ; campoN ; arrayN})

Parámetro	Tipo		Descripción
inicio	Entero largo	➡	Número de registro seleccionado a partir del cual comenzar la recuperación de datos
fin	Entero largo	➡	Número de registro seleccionado donde termina la recuperación de datos
campo	Campo, Tabla	➡	Campo a utilizar para recuperar los datos o Tabla a utilizar para recuperar los números de registros
array	Array	⬅	Array para recibir los datos o números de registros de los campos

Descripción

SELECTION RANGE TO ARRAY crea uno o más arrays y copia los datos de los campos o números de registro de la selección actual en arrays.

A diferencia de **SELECTION TO ARRAY**, que aplica a la totalidad de la selección actual, **SELECTION RANGE TO ARRAY** sólo aplica al rango de los registros seleccionados especificados por los parámetros *inicio y fin*.

El comando espera que los números de registro seleccionados pasados en *inicio* y *fin* cumplan con la fórmula *1 <= inicio <= fin <= Records in selection ([...])*.

Si pasa *1 <= inicio = fin < Records in selection ([...])*, se cargarán los campos u obtendrá el número de registro del registro cuyo registro seleccionado es *inicio = fin*.

Si pasa números de registros seleccionados incorrectos, el comando hace lo siguiente:

- Si *fin > Records in selection ([...])*, devuelve los valores a partir del registro seleccionado especificado por *inicio* hasta el último registro seleccionado.
- Si *inicio > fin*, devuelve los valores del registro cuyo registro seleccionado es *inicio* únicamente.
- Si ambos parámetros son inconsistentes con el tamaño de la selección, devuelve arrays vacíos.

Como **SELECTION TO ARRAY**, el comando **SELECTION RANGE TO ARRAY** aplica a la selección de la tabla especificada en el primer parámetro.

Al igual que **SELECTION TO ARRAY**, **SELECTION RANGE TO ARRAY** también puede realizar las siguientes operaciones:

- Cargar los valores de uno o varios campos.
- Cargar los números de registros utilizando la sintaxis *...;[tabla];Array;...*
- Cargar valores de campos relacionados, si existe una relación automática Muchos a Uno entre las tablas o si usted previamente ha llamado **SET AUTOMATIC RELATIONS** para cambiar las relaciones Muchos a Uno manuales a automáticas. En ambos casos, los valores pueden cargarse a través de varios niveles de relaciones Muchos a Uno entre tablas.

Cada array se digita de acuerdo al tipo de campo.

Cuando aplica **SELECTION RANGE TO ARRAY** a un campo de tipo Hora, note que creará un array de tipo Hora únicamente si el array no se ha definido como de otro tipo. Por ejemplo, en el siguiente contexto, el array *myArray* conservará el tipo Entero largo:

```
ARRAY LONGINT(myArray;0)
SELECTION TO ARRAY([myTable]myTimeField;myArray)
```

Si carga los números de los registros, se copian en un array de tipo Entero largo.

Nota: puede llamar al comando **SELECTION RANGE TO ARRAY** con sólo los parámetros *inicio* y *fin*. Utilice esta sintaxis especial para lanzar, en una selección limitada la ejecución de una serie diferida de comandos **SELECTION TO ARRAY** utilizando el parámetro * (ver ejemplo 4]).

4D Server: **SELECTION RANGE TO ARRAY** es optimizado por 4D Server. Cada array se crea en el servidor y luego se envía, en su totalidad, al equipo cliente.

Advertencia: SELECTION RANGE TO ARRAY puede crear arrays grandes, dependiendo del rango definido en *inicio* y *fin,* y en el tipo y tamaño de los datos a cargar. Los arrays residen en memoria, de manera que es buena idea probar el resultado después de la ejecución del comando. Para hacerlo, pruebe el tamaño de cada array resultante o cubra la llamada al comando, utilizando un método de proyecto **ON ERR CALL**.

Si el comando se ejecuta correctamente, el tamaño de cada array resultante es igual a *(fin-inicio)+1*, excepto si el parámetro *fin* es superior al número de registros en la selección. En tal caso, cada array resultante contiene *(Registros en selección([...])-inicio)+1* elementos.

Ejemplo 1

La siguiente línea de código utiliza los 50 primeros registros de la selección actual de la tabla *[Facturas].* Se cargan los valores del campo *[Facturas]Facturas ID* y del campo relacionado *[Clientes]Clientes ID*.

```
SELECTION RANGE TO ARRAY(1;50;[Facturas]Facturas ID;allnvoID;[Clientes]Clientes ID;alCustID)
```

Ejemplo 2

Las siguientes líneas de código utilizan los 50 primeros registros de la selección actual de la tabla *[Facturas]*. Se cargan los números de registro de la tabla *[Facturas]* así como los de la tabla asociada *[Clientes]*:

```
lSelTalla:=Records in selection([Facturas])
SELECTION RANGE TO ARRAY(lSelTalla-49;lSelTalla;[Facturas];alFacRegN;[Facturas];alCliRegN)
```

Ejemplo 3

Las siguiente líneas de código permiten trabajar secuencialmente en porciones de 1 000 registros de una selección grande que no puede descargase en su totalidad en arrays:

```
lMaxPag:=1000
lSelTalla:=Records in selection([Directorio Telefonico])
For($lPage ;1;1+((lSelTalla-1)\lMaxPag))
  ` Cargar los valores y/o los números de registros
   SELECTION RANGE TO ARRAY(1+(lMaxPag*($lPag-1));lMaxPag*$lPag;...;...;...;...;...;...)
  ` Hacer algo con los arrays
End for
```

Ejemplo 4

Uso de los 50 primeros registros actuales de la tabla [Facturas] para cargar varios arrays, en ejecución diferida:

```
 // Instrucciones diferidas
SELECTION TO ARRAY([Facturas]InvoiceRef;arrLInvRef;*)
SELECTION TO ARRAY([Facturas]Date;arrDInvDate;*)
SELECTION TO ARRAY([Clientes]ClientRef;arrLClientRef;*)
 // Ejecución de las instrucciones diferidas
SELECTION RANGE TO ARRAY(1;50)
```

SELECTION TO ARRAY

SELECTION TO ARRAY {(campo ; array {; campo ; array {; campo2 ; array2 ; ... ; campoN ; arrayN}}{; *})}

Parámetro	Tipo		Descripción
campo	Campo, Tabla	➡	Campo a utilizar para recuperar datos o Tabla a utilizar para recuperar números de registros
array	Array	⬅	Array para recibir valores de campos o números de registros
campo	Campo	➡	Campo a recuperar en el array
array	Array	⬅	Array que recibe los valores del campo
*	Operador	➡	Esperar ejecución

Descripción

El comando **SELECTION TO ARRAY** crea uno o más arrays y copia los valores en los campos o los números de registro de la selección actual en los arrays.

El comando **SELECTION TO ARRAY** se aplica a la selección actual de la tabla designada por el primer parámetro (nombre de tabla o nombre de campo). **SELECTION TO ARRAY**, puede realizar las siguientes operaciones:

- Cargar los valores de uno o varios campos.
- Cargar los números de los registros utilizando la sintaxis *[tabla];array*
- Cargar los valores de los campos relacionados, si hay una relación automática Muchos a Uno entre las tablas o que haya llamado previamente el comando **SET AUTOMATIC RELATIONS** para hacer automáticas las relaciones manuales Muchos a Uno. En ambos caso, los valores se cargan de tablas a través de varios niveles de relaciones Muchos a Uno.

Cada array es definido de acuerdo al tipo de campo.

Cuando aplica **SELECTION TO ARRAY** a un campo de tipo Hora, es importante notar que sólo crea un array de tipo Hora si el array no se ha definido como de otro tipo. Por ejemplo, en el siguiente contexto, el array *myArray* permanece como un array de tipo Entero largo:

```
ARRAY LONGINT(myArray;0)
SELECTION TO ARRAY([myTable]myTimeField;myArray)
```

Si carga los números de registro, se copian en un array de tipo *Entero largo*.

Si pasa el parámetro *, 4D no ejecuta inmediatamente la línea de instrucción correspondiente pero la guarda en memoria; de esta forma puede apilar varias líneas que terminen en *. El conjunto de las líneas en espera es ejecutado por una instrucción **SELECTION TO ARRAY** final sin parámetro *. Por esta razón, ahora el comando puede llamarse sin parámetros. En este caso, los tipos de arrays se verifican al momento de la ejecución de la línea final (sin el parámetro *).
Como para el comando **QUERY**, este principio permite romper una instrucción compleja en un conjunto de líneas, lo cual es más fácil de leer y mantener. También es posible insertar instrucciones intermedias o crear un array dentro de un bucle (ver el ejemplo 2 del comando **ARRAY TO SELECTION**).
4D Server: el comando **SELECTION TO ARRAY** se optimiza para 4D Server. Cada array se crea en el servidor y luego se envía, en su totalidad, al equipo cliente.

Advertencia: el comando **SELECTION TO ARRAY** puede crear arrays grandes, dependiendo del tamaño de la selección actual y del tamaño de los datos a cargar. Los arrays residen en memoria, de tal manera que es una buena idea probar el resultado después de la ejecución del comando. Para hacer esto, pruebe el tamaño de cada array resultante o utilice un método de proyecto **ON ERR CALL**.

Nota: después de un llamado a **SELECTION TO ARRAY**, la selección y el registro actual no se modifican, pero el registro actual no se carga. Si necesita utilizar los valores de los campos del registro actual, utilice el comando **LOAD RECORD** después del comando **SELECTION TO ARRAY**.

Ejemplo 1

En el siguiente ejemplo, la tabla *[Personas]* tiene una relación automática con la tabla [Empresas]. Los dos arrays *asApellido* y *asEmpresaDir* son dimensionados de acuerdo al número de registros seleccionados en la tabla *[Personas]* y contienen la información de ambas tablas:

```
SELECTION TO ARRAY([Personas]Apellido;asApellido;[Empresa]Direccion;asEmpresaDir)
```

Ejemplo 2

El siguiente ejemplo devuelve el número de registros de la tabla *[Clientes]* en el array *alNumerosRegistros* y los valores del campo *[Clientes]Nombres* en el array *asNombres*:

```
SELECTION TO ARRAY([Clientes];alNumerosRegistros;[Clientes]Nombres;asNombres)
```

El mismo ejemplo puede escribirse:

```
SELECTION TO ARRAY([Clientes];alNumerosRegistros;*)
SELECTION TO ARRAY([Clientes]Nombres;asNombres;*)
SELECTION TO ARRAY
```

Size of array

Size of array (array) -> Resultado

Parámetro	Tipo		Descripción
array	Array	➡	Array cuyo tamaño se devuelve
Resultado	Entero largo	⮌	Devuelve el número de elementos en el array

Descripción

El comando **Size of Array** devuelve el número de elementos de *array*.

Ejemplo 1

El siguiente ejemplo devuelve el tamaño del array *anArray*:

```
vlTalla:=Size of array(anArray)  ` vlTalla recibe el tamaño de anArray
```

Ejemplo 2

El siguiente ejemplo devuelve el número de filas en un array bidimensional:

```
vlFilas:=Size of array(a2DArray)  ` vlFilas recibe el tamaño de a2DArray
```

Ejemplo 3

El siguiente ejemplo devuelve el número de columnas de una fila en un array bidimensional:

```
vlColumnas:=Size of array(a2DArray{10})  ` vlColumnas recibe el tamaño de a2DArray{10}
```

SORT ARRAY

SORT ARRAY (array {; array2 ; ... ; arrayN}{; > o <})

Parámetro	Tipo		Descripción
array	Array	➡	Arrays a ordenar
> o <	Operador	➡	">" ordenar en orden ascendente, u "<" ordenar en orden descendente, u orden ascendente si se omite

Descripción

El comando **SORT ARRAY** ordena uno o más array en orden ascendente o descendente.

Nota: no es posible ordenar arrays de tipo *Puntero* o *Imagen*. Puede ordenar los elementos de un array bidimensional (es decir, *a2DArray{$vlEsteElem}*) pero no puede ordenar el array bidimensional en sí mismo (es decir, *a2DArray*).

El último parámetro especifica si ordenar el *array* en orden ascendente o descendente. El símbolo "mayor que" (>) indica un orden ascendente; el símbolo "menor que" (<) indica un orden descendente. Si no especifica el orden, la ordenación es ascendente.

Si se especifica más de un array, los arrays se ordenan siguiendo el definido para el primer array; las ordenaciones multiniveles no son posibles.

En su lugar puede utilizar el comando **MULTI SORT ARRAY** si desea ordenar arrays sincronizados.

Ejemplo 1

El siguiente ejemplo crea dos arrays y luego los ordena en función del nombre de la empresa:

```
ALL RECORDS([Personas])
SELECTION TO ARRAY([Personas]Nombre;asNombres;[Personas]Empresa;asEmpresas)
SORT ARRAY(asEmpresas;asNombres;>)
```

Sin embargo, como **SORT ARRAY** no realiza ordenaciones multinivel, los nombres de las personas aparecen en desorden al interior de cada empresa. Para ordenar las personas por nombre para cada empresa, debe escribir:

```
ALL RECORDS([Personas])
ORDER BY([Personas];[Personas]Empresa;>;[Personas]Nombre;>)
SELECTION TO ARRAY([Personas]Nombre;asNombres;[Personas]Empresa;asEmpresas)
```

Ejemplo 2

Usted visualiza los nombres de una tabla *[Personas]* en una ventana flotante. Cuando hace clic en los botones en la ventana, puede ordenar esta lista de nombres de A a Z o de Z a A. Como varias personas pueden tener el mismo nombre, también puede utilizar un campo *[Personas]Numero ID*, el cual es un campo indexado único. Cuando hace clic en un nombre de la lista, usted recupera el registro para el nombre correspondiente. Manteniendo un array sincronizado y oculto de números de identificación, se asegura de acceder al registro correspondiente al nombre seleccionado:

```
  ` Método de objeto del array asNombres
Case of
   :(Form event=On Load)
      ALL RECORDS([Personas])
      SELECTION TO ARRAY([Personas]Nombre;asNombres;[Personas]Numero ID;todosIDs)
      SORT ARRAY(asNombres;todosIDs;>)
   :(Form event=On Unload)
      CLEAR VARIABLE(asNombres)
      CLEAR VARIABLE(todosiDs)
   :(Form event=On Clicked)
      If(asNombres#0)
  ` Utilice el array todosIDs obtener el registro correcto
         QUERY([Personas];[Personas]Numero ID=todosIDs{asNombres})
  ` Hacer algo con el registro
      End if
End case

  ` Método de objeto del botón bA2Z
  ` Ordenación de los arrays en orden creciente conservando la sincronización
SORT ARRAY(asNombres;todosIDs;>)

  ` Método de objeto del botón bZ2A
  ` Ordenación de los arrays en orden decreciente conservando la sincronización
SORT ARRAY(asNombres;todosIDs;<)
```

TEXT TO ARRAY

TEXT TO ARRAY (varText ; arrText ; ancho ; nomFuente ; tamFuente {; estiloFuente {; *}})

Parámetro	Tipo		Descripción
varText	Texto	➡	Texto original a dividir
arrText	Array texto	⬅	Array que contiene el texto dividido en palabras o líneas
ancho	Entero largo	➡	Ancho máximo de la cadena(en píxeles)
nomFuente	Texto	➡	Nombre de la fuente
tamFuente	Entero largo	➡	Tamaño de la fuente
estiloFuente	Entero largo	➡	Estilo de fuente
*	Operador	➡	Si se pasa = interpretar el texto como multistyle

Descripción

El comando **TEXT TO ARRAY** transforma una variable texto en un array texto. El texto original (con estilo o no) se divide y cada parte se convierte en un elemento del array *arrText* que es devuelto por el comando. Este comando se puede utilizar por ejemplo para llenar las páginas o las columnas con texto de un tamaño fijo.

El texto original se divide en "palabras", basado en un tamaño de línea definido por los parámetros del comando y teniendo en cuenta todos los estilos utilizados.

En el parámetro *varText*, pase el texto a dividir en elementos de array. Este texto puede ser o no multiestilo. Algunos parámetros se ignoran cuando el texto es multiestilo.

Pase en el parámetro *arrText* el nombre del array a llenar con el texto dividido.

En el parámetro *ancho*, pase un tamaño en píxeles que indique la longitud máxima de línea a medir para dividir el texto. Para todo el texto, el comando evaluara el número máximo de palabras que pueden "encajar" en este ancho en función de los atributos gráficos del texto (fuente, estilo).

- Si el texto es multiestilo, los estilos del texto original se tienen en cuenta y los siguientes parámetros se ignoran si se pasan. En este caso, las líneas de texto en el array resultante conservan su estilo original (de manera que se puede imprimir una por una vía una variable texto o alfa por ejemplo).
- Si se trata de texto plano (sin estilos), debe pasar todos los parámetros para que el comando pueda calcular la longitud de las líneas.

Cada elemento del array debe contener al menos una palabra. Si el ancho pasado es demasiado pequeño para que la regla de división se respete estrictamente, el array se llena lo más aproximadamente posible de acuerdo a los parámetros y la variable OK toma el valor 0. Por ejemplo, si pasa un ancho de 3 píxeles, es probable que la mayoría de las palabras sean más grandes que esta longitud. En este caso, la variable OK toma el valor 0.
Esto también significa que el tamaño máximo teórico del array devuelto es igual al número de palabras presentes en *varText*.

En los parámetros *nomFuente* y *tamFuente*, pase el nombre y el tamaño de la fuente con la cual *varText* debe ser evaluado por el comando para hacer la división. Estos parámetros son obligatorios en el caso de texto sin formato.

En el parámetro *estiloFuente*, pase una o más constantes del tema **Estilos de fuente**.

Constante	Tipo	Valor
Bold	Entero largo	1
Italic	Entero largo	2
Plain	Entero largo	0
Underline	Entero largo	4

Este parámetro es opcional; cuando se omite, se utiliza el estilo Normal.

El parámetro opcional *, si se pasa, permite forzar el que se tenga en cuenta los parámetros *nomFuente*, *tamFuente* y/o *estiloFuente* para los textos multiestilos cuando estos parámetros no están definidos en el texto original. Sin embargo, si estos parámetros están definidos en el texto original, los parámetros pasados al comando se ignoran en todos los casos.

Ejemplo 1

Queremos dividir un texto multistyle en líneas con un tamaño máximo de 200 píxeles:

```
TEXT TO ARRAY(theText;TextArray;200;"Arial";20;Normal;*)
 // los atributos Arial, 20 y Normal sólo se tienen en cuenta si no están definidos en el texto
```

Ejemplo 2

Queremos dividir un texto en líneas de un tamaño máximo de 350 píxeles en fuente Bodoni negrita 14. Como el comando no funciona correctamente si la fuente no está disponible, es útil verificar su presencia:

```
ARRAY TEXT($FontList;0)
FONT LIST($FontList)
$Font:="Bodoni"
$p:=Find in array($FontList;$Font)
If($p>0)
    TEXT TO ARRAY(theText;TextArray;350;"Bodoni";14;Bold)
Else
 // utilizar otra fuente
End if
```

Ejemplo 3

Un texto multiestilo debe imprimirse sin estilo en la fuente Arial normal 12 con un ancho máximo de 600 píxeles:

```
 // transformamos el texto multiestilo en texto bruto
$RawText:=OBJECT Get plain text(vText)
 // llenamos el array
TEXT TO ARRAY($RawText;TextArray;600;"Arial";12)
```

Ejemplo 4

Debe imprimir en un área de 400 píxeles de largo un texto de un máximo de 80 líneas con la fuente más grande posible (sin exceder los 24 puntos). Puede escribir:

```
ARRAY TEXT(TextArray;0)
$Size:=24
Repeat
    TEXT TO ARRAY($RawText;TextArray;400;"Arial";$Size)
    $Size:=$Size-1
    $n:=Size of array(TextArray)
Until($n<=80)
```

_o_ARRAY STRING

_o_ARRAY STRING (strLon ; nombreArray ; tamaño {; tamaño2})

Parámetro	Tipo		Descripción
strLon	Entero largo	➡	Longitud de la cadena (1... 255)
nombreArray	Array	➡	Nombre del array
tamaño	Entero largo	➡	Número de elementos en el array o Número de filas si se especifica tamaño2
tamaño2	Entero largo	➡	Número de columnas en un array bidimensional

Nota de compatibilidad

El funcionamiento del comando **_o_ARRAY STRING** es estrictamente idéntico al del comando **ARRAY TEXT** (el parámetro *strLen* se ignora).
Actualmente se recomienda utilizar **ARRAY TEXT** exclusivamente en sus desarrollos 4D.

Backup

- Método de base de datos On Backup Shutdown
- Método de base de datos On Backup Startup
- BACKUP
- CHECK LOG FILE
- GET BACKUP INFORMATION
- GET RESTORE INFORMATION
- INTEGRATE MIRROR LOG FILE
- Log File
- LOG FILE TO JSON
- New log file
- RESTORE
- SELECT LOG FILE
- *_o_INTEGRATE LOG FILE*

Método de base de datos On Backup Shutdown

El **Método de base de datos On Backup Shutdown** se llama cada vez que termina un backup de la base. Las razones para detener un backup pueden ser el fin de la copia, la interrupción por parte del usuario o un error.
Esto concierne a todos los entornos 4D (todos los modos), 4D Server así como las aplicaciones 4D compiladas y fusionadas con 4D Volume Desktop.

El **Método de base de datos On Backup Shutdown** permite verificar que el backup fue ejecutado correctamente. El método recibe, en el parámetro *$1*, un valor indicando el estado del backup una vez terminado:

- Si el backup se ejecutó correctamente, *$1* es igual a 0.
- Si el backup fue interrumpido por el usuario o por un error, *$1* es diferente de 0.
    - Si el backup fue detenido por el **Método de base de datos On Backup Startup** (*$0* # 0), *$1* obtiene el valor devuelto en el parámetro *$0*. Esto le permite implementar un sistema de gestión de errores personalizado.
    - Si el backup fue detenido por un error, el código del error se devuelve en *$1*.

En todos los casos, puede obtener información sobre el error utilizando el comando **GET BACKUP INFORMATION**.

Nota: debe declarar el parámetro *$1* (entero largo) en el método de la base:

```
C_LONGINT($1)
```

Es importante notar que en caso de error durante el backup (disco lleno, soporte inaccesible, etc.), la información relativa al error se muestra únicamente en el monitor de 4D Server o en el CSM, y se copia en el historial de backups. No se muestra una caja de diálogo de alerta y la variable *error* no se modifica. Si quiere notificar al administrador que se produjo un error, particularmente en el contexto de una aplicación en modo cliente/servidor, es necesario utilizar el **Método de base de datos On Backup Shutdown**.

Método de base de datos On Backup Startup

El **Método de base de datos On Backup Startup** se llama cada vez que un backup está a punto de iniciar (backup manual, backup automático programado, o utilizando el comando **BACKUP**).

Esto concierne a todos los entornos 4D: 4D en modo local, 4D Server, 4D en modo remoto, 4D Desktop y bases fusionadas con 4D Desktop.

El **GET BACKUP INFORMATION** permite verificar el inicio del backup. En este método, debe devolver en el parámetro $0 un valor que autorice o rechace el backup:

- Si $0 = 0, el backup puede comenzar.
- Si $0 # 0, el backup no es autorizado. La operación se cancela y devuelve un error. Puede obtener el error utilizando el comando **Método de base de datos On Backup Startup**.

Puede utilizar este método base para verificar las condiciones de ejecución del backup (usuario, fecha del último, etc.).

Nota: debe declarar el parámetro *$0* (entero largo) en el método de la base:

```
C_LONGINT($0).
```

BACKUP

BACKUP
Este comando no requiere parámetros

Descripción

El comando **BACKUP** inicia el backup de la base de datos utilizando los parámetros de copia de seguridad actuales. No aparece una caja de diálogo de confirmación; sin embargo, aparece una barra de progreso en la pantalla.

Los parámetros de backup se definen en las Propiedades de la base ("Preferencias" en 4D v11 SQL). Igualmente son almacenados en el archivo Backup.XML ubicado en la subcarpeta Preferences/Backup de la base de datos.

El comando **BACKUP** llama al **Método de base de datos On Backup Startup** al comienzo de su ejecución y al **Método de base de datos On Backup Shutdown** al final de su ejecución.

Por este mecanismo, el comando no debe llamarse desde uno de estos métodos base.

4D Server: cuando se llama desde un equipo cliente, el comando **BACKUP** se considera como un procedimiento almacenado; siempre ejecutado en el servidor.

Variables y conjuntos del sistema

Si el backup se realiza correctamente, la variable del sistema OK toma el valor 1; de lo contrario, toma el valor 0.

Gestión de errores

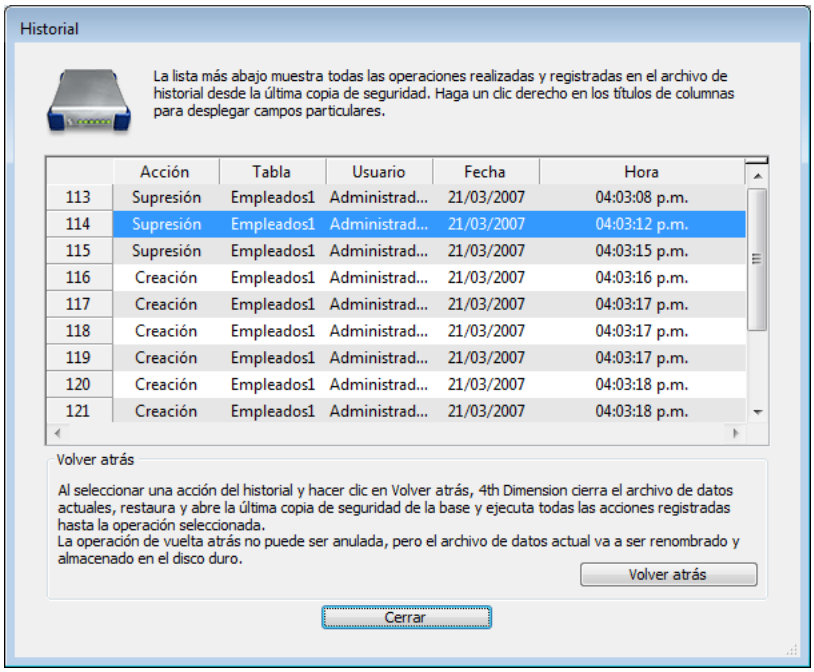
En caso de que se presenten incidentes durante el backup, la información relativa al incidente se escribe en el diario de backup y el error de más alto nivel se envía únicamente al **Método base On Backup Shutdown**. Por lo tanto es importante utilizar este método base para poder administrar por programación los errores relacionados con el backup.

CHECK LOG FILE

CHECK LOG FILE
Este comando no requiere parámetros

Descripción

El comando **CHECK LOG FILE** muestra la caja de diálogo de visulización del archivo de historial actual de la base (accesible también vía la ventana del Centro de seguridad y mantenimiento) :



Esta caja de diálogo incluye el botón **Volver atrás** que permite cancelar las operaciones efectuadas a los datos de la base. Para mayor información sobre esta caja de diálogo, por favor consulte el Manual de Diseño.

Nota: como la función Volver atrás es relativamente poderosa, es recomendable restringir el acceso al comando **CHECK LOG FILE** de los administradores de la base.

Este comando sólo puede utilizarse en el contexto de aplicaciones monousuario. Más particularmente, permite acceder a la función volver atrás desde las aplicaciones 4D Volume Desktop (aplicaciones sin modo diseño). Si se llama en una aplicación cliente/servidor, el comando no tiene efecto y devuelve el error 1421.

Manejo de errores

- Si este comando se ejecuta en una base de datos que funciona sin archivo de historial, el comando no hace nada y devuelve el error 1403.
- Si este comando se ejecuta en una base cliente/servidor, el comando no hace nada y devuelve el error 1421.
  Puede interceptar estos errores utilizando un método de manejo de errores instalado por el comando **ON ERR CALL**.

GET BACKUP INFORMATION

GET BACKUP INFORMATION (selector ; info1 ; info2)

Parámetro	Tipo		Descripción
selector	Entero largo	➡	Tipo de información a obtener
info1	Entero largo, Fecha	⬅	Valor 1 del selector
info2	Hora, Cadena	⬅	Valor 2 del selector

Descripción

El comando **GET BACKUP INFORMATION** permite obtener información relacionada con el último backup efectuado en la base de datos.

Pase el tipo de información a obtener en *selector*. Puede utilizar una de las siguientes constantes, ubicadas en el tema "**Backup and Restore**":

Constante	Tipo	Valor
Last backup date	Entero largo	0
Last backup status	Entero largo	2
Next backup date	Entero largo	4

El tipo y el contenido de los parámetros *info1* e *info2* depende del valor del *selector*.
- Si *selector* = 0 (Last Backup Date), *info1* devuelve la fecha e *info2* la hora del último backup.
- Si *selector* = 2 (Last Backup Status), *info1* devuelve el número e *info2* el texto del estado del último backup.
- Si *selector* = 4 (Next Backup Date), *info1* devuelve la fecha e *info2* la hora del próximo backup programado.

GET RESTORE INFORMATION

GET RESTORE INFORMATION (selector ; info1 ; info2)

Parámetro	Tipo		Descripción
selector	Entero largo	➡	Tipo de información a recuperar
info1	Entero largo, Fecha	⬅	Valor 1 del selector
info2	Cadena, Hora	⬅	Valor 2 del selector

Descripción

El comando **GET RESTORE INFORMATION** permite obtener información relacionada con la última restauración automática de la base.

Pase el tipo de información a obtener en *selector*. Puede utilizar una de las siguientes constantes, ubicadas en el tema "**Backup and Restore**":

Constante	Tipo	Valor
Last restore date	Entero largo	0
Last restore status	Entero largo	2

El tipo y el contenido de los parámetros *info1* e *info2* dependen del valor de *selector*.
• Si *selector* = 0 (Last Restore Date), *info1* devuelve la fecha e *info2* la hora de la última restauración automática de la base.
• Si *selector* = 2 (Last Restore Status), *info1* devuelve el número e *info2* el texto del estado de la última restauración automática de la base.

Nota: este comando no tiene en cuenta restauraciones manuales de la base.

INTEGRATE MIRROR LOG FILE

INTEGRATE MIRROR LOG FILE (rutaAcceso ; numOperacion {; modo {; objError}})

Parámetro	Tipo		Descripción
rutaAcceso	Texto	➡	Nombre o ruta de acceso del archivo de historial a integrar
numOperacion	Real variable	➡	Número de la última operación integrada
		⬅	Nuevo número de la última operación integrada o -2 para integrar todo el archivo
modo	Entero largo	➡	0=modo estricto (modo por defecto), 1=modo auto reparar
objError	Object variable	⬅	Operaciones faltantes

Descripción

Nota preliminar: este comando sólo funciona con 4D Server. Únicamente puede ejecutarse vía el comando **Execute on server** o en un procedimiento almacenado.

El **INTEGRATE MIRROR LOG FILE** permite integrar el archivo de historial designado por *rutaAcceso* en una base 4D Server, a partir de la operación *numOperacion*.

El comando acepta integrar todo archivo de historial en la base, incluso si no corresponde al archivo de datos. Este comando está destinado específicamente para su uso en el contexto de una base espejo.

Nota: a partir de 4D v14, es posible utilizar un archivo de historial en el contexto de una base "espejo": la opción "Utilizar archivo de historial" ahora se puede seleccionar en las propiedades de una base 4D Server utilizada como espejo lógico, por tanto, permitiendo la implementación de servidores espejo en serie (ver la sección **Configurar un espejo lógico** en el manual 4D Server).

A diferencia del comando **_o_INTEGRATE LOG FILE**, el comando **INTEGRATE MIRROR LOG FILE** no sustituye el archivo de historial integrado al historial actual al ejecutarse: el archivo de historial de la base continua siendo utilizado. En consecuencia, cualquier cambio realizado durante la integración se guarda en el archivo de historial actual.

En *rutaAcceso*, pase una ruta absoluta o relativa a la carpeta de la base. Si pasa una cadena vacía en este parámetro, aparecerá una caja de diálogo estándar de apertura de archivo para que pueda designar el archivo a integrar. Si se cancela esta caja de diálogo, ningún archivo se integra y la variable sistema *OK* toma el valor 0.

En la variable *numOperacion*, pase el número de la última operación integrada, de manera que la integración comience en la siguiente operación. Después de la integración, el valor de la variable *numOperacion*, se actualiza con el número de la última operación integrada. Esto le permite seguir adelante con posteriores integraciones del archivo de historial, utilizando **[#current_title]**. Pase -2 en la variable a integrar todas las operaciones en el archivo de historial.

Nota de compatibilidad: *en las versiones de 4D anteriores a la v15 R4, el parámetro numOperacion era opcional; Sin embargo, desde ahora en adelante, si se omite el parámetro operationNum, se genera un error. Para restablecer el funcionamiento original de su antiguo código, puede simplemente pasar -2 en el parámetro numOperacion.*

En *modo*, pase el modo de integración a activar. Puede utilizar una de las siguientes constantes que se encuentran en el tema "**Backup**":

Constante	Tipo	Valor	Comentario
Auto repair mode	Entero largo	1	Utilizar el modo flexible con las acciones de reparación automática y llenar el parámetro *errObject* (si lo hay)
Strict mode	Entero largo	0	Utilice el modo de integración estricto (por defecto)

- Strict mode: en este modo, tan pronto como se produce un error durante la integración, se detiene y hay que utilizar el CSM para rastrear el error. Este modo de seguridad se utiliza de forma predeterminada y se recomienda en la mayoría de los casos.
- Auto repair mode: en este modo, cuando un error no crítico se produce, se omite y continúa la integración. Si pasa el parámetro *objError*, cada error se registra y se pueden analizar después.
  Los casos de errores no críticos son:
  - El archivo de historial solicita agregar un registro, pero este registro ya existe en los datos.
    Acción de reparación: 4D actualiza el registro.
  - El archivo de historial solicita actualizar un registro, pero aún no existe este registro.
    Acción de reparación: 4D añade el registro.
  - El archivo de historial solicita eliminar un registro, pero no existe este registro.
    Acción de reparación: 4D no hace nada.

Nota: en modo estricto (modo por defecto), la integración se detendrá en el primer error encontrado. En este caso, si desea continuar con la integración tendrá que utilizar el CSM.

Cuando se produce una de las anomalías en modo auto reparación, el registro en cuestión queda automáticamente "reparado" y la operación relacionada se registra en el parámetro *objError*.
Una vez finalizada la ejecución, el parámetro *objError* lista todos los registros reparados. Contiene un único array de objetos denominado "operaciones" construido de la siguiente manera:

```
{"operations":
    [
        {
            "operationType":24,
            "operationName":"Create record",
            "operationNumber":2,
            "contextID":48,
```

```
            "timeStamp":"2015-07-10T07:53:02.413Z",
            "dataLen":24,
            "recordNumber":0,
            "tableID":"F4CXXXXX",
            "tableName":"Customers",
            "fields": {
                  "1": 9,
                  "2": "test value",
                  "3": "2003-03-03T00:00:00.000Z",
                  "4": "BlobPath: Table 1/Field 4/Data_9ACB28F1A2744FDFA5822B22F18B2E12.png",
                  "8": "BlobID: 2"
               }
      },
      {...}
]
```


Atención: el modo de reparación auto debe ser utilizado en casos específicos, ya que pasa por alto las funcionalidades de comprobación de integridad de datos internas de 4D. Se puede utilizar, por ejemplo, cuando un archivo de registro intermedio se ha perdido o dañado y que desea recuperar tantas operaciones como sea posible. En cualquier caso, es necesario prestar atención a la integridad de los datos específicos al utilizar este modo.

La lista actual de propiedades disponibles depende del tipo de operación (por ejemplo: crear registros, eliminar registros, modificar registros, etc.). Estas son las principales propiedades:

- *tipoOperacion*: código interno para la operación
- *nomOperacion*: tipo de operación, por ejemplo, "crear registro", "modificar registro"
- *numOperacion*: número interno de la operación en el archivo de registro
- *contextID*: ID del contexto de ejecución
- *timeStamp*: timestamp de la operación en el archivo de registro
- *dataLong*: tamaño de los datos
- *numRegistro*: número de registro interno
- *IDtabla*: ID interno de la tabla
- *nomTabla*: nombre de la tabla
- *campos*: de valores de campo
- *numSecuencia*: número de secuencia.

Ejemplo

Usted desea integrar un archivo de historial en el servidor espejo en modo de reparación auto:

```
 //a ejecutar en el servidor
C_OBJECT($err)
C_LONGINT($num)  //--2 para integrar todas las operaciones
INTEGRATE MIRROR LOG FILE("c:\\mirror\\logNew.journal";$num;Auto repair mode;$err)
```

Variables y conjuntos del sistema

Si la integración se efectúa correctamente, la variable sistema OK toma el valor 1; de lo contrario, toma el valor 0.

Log File

Log File -> Resultado

Parámetro	Tipo		Descripción
Resultado	Cadena	➲	Nombre completo del archivo historial de la base

Descripción

El comando **Log File** devuelve el nombre largo (es decir la ruta de acceso completa del archivo, incluyendo su nombre) del archivo historial actual de la base abierta.

Si la base funciona sin un archivo historial, el comando devuelve una cadena vacía y la variable sistema OK toma el valor 0.

Si la base funciona con un archivo historial, la variable sistema OK toma el valor 1. La ruta de acceso devuelta por el comando se expresa con la sintaxis de la plataforma actual.

Advertencia: si ejecuta este comando desde un equipo 4D Client, sólo devuelve el nombre del archivo historial, no el nombre largo.

Variables y conjuntos del sistema

- Si la base funciona sin archivo de historial, la variable sistema OK toma el valor 0; de lo contrario, toma el valor 1.
- Si por alguna razón el archivo de historial se vuelve inaccesible durante la sesión de trabajo, se genera el error 1274 y 4D Server no permitirá a los usuarios modificar o escribir datos. Cuando el archivo de historial se vuelve accesible de nuevo, es necesario hacer un backup.

LOG FILE TO JSON

LOG FILE TO JSON (rutaCarpetaDest {; tamMax {; rutaHist {; atribCampo}}})

Parámetro	Tipo		Descripción
rutaCarpetaDest	Texto	➡	Ruta de acceso de la carpeta de destino del archivo guardado
tamMax	Entero largo	➡	Tamaño máximo del archivo JSON a crear (bytes)
rutaHist	Texto	➡	Ruta de acceso del archivo de historial a exportar; utilizar el historial actual si se omite
atribCampo	Entero largo	➡	Atributo de descripción del campo: 1 = utilizar número(por defecto), 2 = utilizar nombre

Descripción

El comando **LOG FILE TO JSON** guarda en formato JSON el archivo de historial actual o todo archivo de historial especificado.

Una vez un historial (archivo binario) se guarda en JSON, su contenido puede ser leído e interpretado por el administrador de la base o por cualquier usuario con el fin de analizar los eventos de la base, por ejemplo.

En *rutaCarpetaDest*, pase la ruta de la carpeta en la que desea almacenar el archivo JSON. Este archivo se llama **JournalExport.json**.

De forma predeterminada, el tamaño máximo del archivo JSON exportada es de 10 MB. Cuando se alcanza este tamaño, el archivo se cierra y se crea un nuevo archivo. Limitar el tamaño de cada archivo JSON reduce los requerimientoos de memoria para el análisis de los archivos. Puede cambiar el tamaño máximo para el archivo exportado definiendo un valor (en bytes) en el parámetro *tamMax*. Pasar 0 restaura el tamaño predeterminado. Pasar un valor negativo elimina todo límite de tamaño.

Por defecto, si se omite el parámetro *rutaHist*, el comando guarda el archivo de historial actual. Si desea exportar un archivo de historial específico, pase su ruta en el parámetro *rutaHist*. El archivo de historial debe ser un archivo con una extensión ".journal". Si desea exportar un archivo de historial archivado (extensión ".4bl"), es necesario convertirlo de antemano con el comando**RESTORE**. Puede pasar una cadena vacía ("") para mostrar el diálogo estándar de abrir archivo, lo que permite al usuario seleccionar el archivo de historial a guardar. La ruta del archivo de historial seleccionado se devuelve en la variable sistema **Document**.

Nota: cuando el comando guarda el archivo de historial actual, la base no está bloqueada. Las nuevas operaciones se pueden ejecutar mientras el archivo se escribe en el disco, estas operaciones no se incluirán en el archivo guardado.

Cuando exporta el archivo de historial actual, el parámetro *atribCampo* le permite definir cómo se describen en el atributo exportado: por número (por defecto), o por nombre. Puede pasar una de las siguientes constantes, que se encuentra en el tema "**Backup**":

Constante	Tipo	Valor	Comentario
Field attribute with name	Entero largo	2	Los campos son identificados por su nombre. Ejemplo: {"Apellido":"Gómez"}
Field attribute with number	Entero largo	1	Los campos se identifican por su número (por defecto si se omite). Ejemplo: { "5":"Jones"}.

Nota: cuando exporta un archivo de historial externo, los campos siempre se identifican por su número.

El archivo JSON guardado contiene todas las operaciones registradas en el historial, en forma de un array de objetos JSON. Cada objeto contiene varias propiedades que describen la operación. Ejemplo:


```
[
   {
      "operationType":25,
      "operationName":"Modify record",
      "operationNumber":45,
      "contextID":37,
      "timeStamp":"2015-06-11T09:13:17.138Z",
      "dataLen":42,
      "recordNumber":4,
      "tableID":"5AFA15123F991C43B6ACF8B46A914BD0",
      "tableName":"elem",
      "fields": {
            "1": "primkey5",
            "2": -5,
            "5": "data 25"
        },
      "primaryKey": "8"
   },
   {
      "operationType":23,
      "operationName":"Save seqnum",
      "operationNumber":46,
      "contextID":37,
      "timeStamp":"2015-06-11T09:13:17.138Z",
      "sequenceNumber":23,
      "tableID":"5AFA15123F991C43B6ACF8B46A914BD0",
      "tableName":"elem"
   },
```

```
    {
        "operationType":24,
        "operationName":"Create record",
        "operationNumber":47,
        "contextID":37,
        "timeStamp":"2015-06-11T09:13:17.138Z",
        "dataLen":570,
        "recordNumber":7,
        "tableID":"5AFA15123F991C43B6ACF8B46A914BD0",
        "tableName":"elem",
        "fields": {
                "1": 9,
                "2": "test value",
                "3": "2003-03-03T00:00:00.000Z",
                "4": "BlobPath: Table 1/Field 4/Data_9ACB28F1A2744FDFA5822B22F18B2E12.png",
                "8": "BlobID: 2"
        },
        "extraData": {
                "task_id": 1,
                "user_name": "Vanessa Smith",
                "user4d_id": 1,
                "host_name": "iMac-VSmith-0833",
                "task_name": "Application process",
                "client_version": -1610541776
        },
        "primaryKey": "9"
    }
]
```


Nota: si pasó Field attribute with name en el parámetro *atribCampo*, el objeto "fields" contendrá:


```
...
        "fields": {
                "ID": 9,
                "Field_2": "test value",
                "Date_Field": "2003-03-03T00:00:00.000Z",
                "Field_4": "BlobPath: Table 1/Field 4/Data_9ACB28F1A2744FDFA5822B22F18B2E12.png",
                "Field_8": "BlobID: 2"
        },...
```


La lista actual de propiedades disponibles depende del tipo de operación (por ejemplo: crear registros, eliminar registros, modificar registros, crear Blob etc.). Estas son las principales propiedades:

- *tipoOperacion*: código interno para la operación
- *nomOperacion*: tipo de operación, por ejemplo, "crear registro", "modificar registro"
- *numOperacion*: número interno de la operación en el archivo de registro
- *contextID*: ID del contexto de ejecución
- *timeStamp*: timestamp de la operación en el archivo de registro
- *dataLong*: tamaño de los datos
- *numRegistro*: número de registro interno
- *IDtabla*: ID interno de la tabla
- *nomTabla*: nombre de la tabla
- *campos*: de valores de campo
- *numSecuencia*: número de secuencia dentro de la secuencia de auto incrementar.

Ejemplo

Usted quiere guardar el archivo de historial actual en JSON:

```
LOG FILE TO JSON("c:\\4Dv15\\ExportLogs")
```

Usted quiere guardar un archivo de historial archivado en JSON:

```
LOG FILE TO JSON("c:\\4Dv15\\ExportLogs";0;"c:\\4Dv15\\Backup\\old_myDB.journal")
```

Variables y conjuntos del sistema

El comando **LOG FILE TO JSON** modifica el valor de las variables sistema OK y Document: si el archivo JSON se guarda correctamente, OK toma el valor 1 y Document contiene la ruta del archivo de historial. Si ha pasa "" en el parámetro *rutaLog* y

el usuario cancela el cuadro de diálogo de selección de archivos, OK toma el valor 0 y el Document contiene una cadena vacía. Si el usuario selecciona un archivo no válido, OK toma el valor 0 y Document contiene la ruta del archivo invalido.

New log file

New log file -> Resultado

Parámetro	Tipo		Descripción
Resultado	Texto		Ruta de acceso completa del archivo historial cerrado

Descripción

Nota preliminar: este comando sólo funciona con 4D Server. Sólo puede ser ejecutado vía el comando **Execute on server** o en un procedimiento almacenado.

El comando **New log file** cierra el archivo historial actual, le da otro nombre y crea uno nuevo con el mismo nombre en la misma ubicación del anterior. Este comando es para definir un sistema de backup por espejo lógico. (ver la sección "**Configurar un espejo lógico**" en el Manual de 4D Server).

El comando devuelve la ruta de acceso completa (ruta de acceso + nombre) del archivo historial cerrado (llamado "segmento"). Este archivo es guardado en la misma ubicación del archivo historial actual (especificada en la página Configuración en el tema Copia de seguridad de las Preferencias). El comando no efectúa ningún proceso (compresión, segmentación) en el archivo guardado. No aparece ninguna caja de diálogo.

El archivo se renombra con los números de backup´actuales de la base y el archivo historial, como se muestra en el siguiente ejemplo: *NombreBase[BackupNum-LogBackupNum].4DL*. Por ejemplo:

- Si la base MiBase.4DD ha sido guardada 4 veces, el último archivo de backup se llamará *MyBase[0004].4BK*. El nombre del primer "segmento" del archivo historial será por lo tanto *MiBase[0004-0001].journal*.
- Si la base MiBase.4DD ha sido guardada 3 veces y el archivo historial ha sigo guardado 5 veces, el nombre del sexto backup del archivo historial será *MiBase[0003-0006].journal*.

Gestión de errores

En caso de que se presente un error, el comando genera un código que puede interceptarse utilizando el comando **ON ERR CALL**.

RESTORE

RESTORE {(rutaArchivo {; rutaCarpetaDest})}

Parámetro	Tipo		Descripción
rutaArchivo	Texto	➡	Ruta de acceso del archivo a restituir
rutaCarpetaDest	Texto	➡	Ruta de acceso de la carpeta de destino

Descripción

El comando **RESTORE** permite restituir el o los archivos incluidos en un archivo 4D. Este comando es útil con interfaces personalizadas para la gestión de backups.

Si no pasa el parámetro *rutaArchivo*, el comando muestra una caja de diálogo de apertura para que el usuario pueda seleccionar el archivo a restaurar.

El parámetro *rutaArchivo* permite indicar la ruta de acceso del archivo a restituir. Esta ruta debe expresarse con la sintaxis del sistema. Puede pasar una ruta absoluta o una relativa al archivo de estructura de la base.
En este caso (si se omite el parámetro *rutaCarpetDest*), la caja de diálogo de restitución estándar aparece con el archivo preseleccionado, de manera que el usuario puede designar la carpeta de destino. Cuando se completa el procedimiento, aparece una caja de diálogo de alerta y se muestra la carpeta que contiene los elementos restituidos.

También puede pasar el parámetro *rutaCarpetDest* con la ruta de acceso de la carpeta de destino de los elementos restituidos. Esta ruta debe expresarse con la sintaxis sistema. Puede pasar una ruta absoluta o relativa al archivo de estructura de la base. Si pasa este parámetro, aparece una caja de diálogo de restauración preconfigurada, permitiendo únicamente al usuario lanzar o cancelar la restitución. Cuando se completa el procedimiento, la ventana se cierra nuevamente sin mostrar información adicional.

El comando **RESTORE** modifica el valor de las variables *OK* y *Document*: si la restitución es correcta, *OK* toma el valor 1 y *Document* contiene la ruta de la carpeta de restitución. Si el usuario cancela la caja de restauración, interrumpe la restauración o si ocurre un error, *OK* toma el valor 0 y *Document* contiene una cadena vacía. Puede interceptar el error utilizando un método instalado vía el comando **ON ERR CALL**.

Nota: en una aplicación 4D compilada y fusionada con 4D Volume Desktop, el comando **RESTORE** produce la visualización de una caja de diálogo estándar de apertura de archivos que lista por defecto los archivos de extensión "4BK".

SELECT LOG FILE

SELECT LOG FILE (historial)

Parámetro	Tipo		Descripción
historial	Operador, Cadena	➡	Nombre del archivo historial o * para cerrar el historial actual

Descripción

El comando **SELECT LOG FILE** crea, o cierra el archivo historial de acuerdo al valor que se pase en *historial*.

Nota: llamar al comando **SELECT LOG FILE** es equivalente a seleccionar/deseleccionar la opción **Utilizar el historial** en la página **Copia de seguridad**/**Configuración** de las Preferencias de la aplicación.

Pase en *historial*, el nombre o la ruta de acceso completa del archivo historial a crear. Si pasa únicamente un nombre, el archivo se creará junto al archivo de estructura de la base.

Si pasa una cadena vacía en *historial*, **SELECT LOG FILE** presenta una caja de diálogo estándar de registro de archivo, permitiendo al usuario elegir el nombre y la ubicación del archivo historial a crear. Si el archivo se crea correctamente, la variable **OK** toma el valor 1. Por el contrario, si el usuario hace clic en el botón Cancelar o si el archivo historial no se puede crear, **OK** toma el valor 0.

Nota: el nuevo archivo de historial no se genera de inmediato después de la ejecución del comando, sino después del siguiente backup (el parámetro se conserva en el archivo de datos y se tendrá en cuenta incluso si la base se cierra). Puede llamar al comando BACKUP para provocar la creación del archivo historial.

Si pasa *"*"* en *historial*, **SELECT LOG FILE** cierra el archivo historial actual de la base. La variable **OK** toma el valor 1 cuando el archivo historial está cerrado.

Si utiliza **SELECT LOG FILE** para crear un archivo historial antes de que el backup haya terminado y el archivo de datos ya contiene registros, 4D genera el error *-4447,* el cual puede interceptar con un método **ON ERR CALL**.

Variables y conjuntos del sistema

OK toma el valor 1 si el archivo histórico es creado o cerrado correctamente.

Gestión de errores

Se genera el error -4447 si no se puede realizar la operación porque la base de datos necesita una copia de seguridad. Puede interceptar el error con un método **ON ERR CALL**.

_o_INTEGRATE LOG FILE

_o_INTEGRATE LOG FILE (rutaAcceso)

Parámetro	Tipo		Descripción
rutaAcceso	Texto	➡	Nombre o ruta de acceso del archivo historial a integrar

Nota de compatibilidad

El comando **_o_INTEGRATE LOG FILE** es declarado obsoleto en 4D a partir de la versión 16 y se mantiene únicamente por razones de compatibilidad. La funcionalidad de copia de seguridad de 4D que utiliza un espejo lógico ahora se basa únicamente en el comando **INTEGRATE MIRROR LOG FILE**, que ha sido optimizado y ofrece una mayor flexibilidad.

BLOB

- Comandos BLOB
- BLOB PROPERTIES
- BLOB size
- BLOB TO DOCUMENT
- BLOB to integer
- BLOB to list
- BLOB to longint
- BLOB to real
- BLOB to text
- BLOB TO VARIABLE
- COMPRESS BLOB
- COPY BLOB
- DECRYPT BLOB
- DELETE FROM BLOB
- DOCUMENT TO BLOB
- ENCRYPT BLOB
- EXPAND BLOB
- INSERT IN BLOB
- INTEGER TO BLOB
- LIST TO BLOB
- LONGINT TO BLOB
- REAL TO BLOB
- SET BLOB SIZE
- TEXT TO BLOB
- VARIABLE TO BLOB

Comandos BLOB

Definición

4D soporta datos de tipo BLOB (Binary Large OBjects).
Puede definir campos y variables de tipo BLOB y arrays de tipo BLOB:

- Para crear un campo de tipo BLOB, seleccione BLOB en la lista desplegable **Tipo de campo** en la ventana **Propiedades del campo**.
- Para crear una variable de tipo BLOB, utilice el comando de declaración de compilación **C_BLOB**. Puede crear variables BLOB locales, proceso e interproceso.
- Para crear un array de tipo BLOB, utilice el comando **ARRAY BLOB**.

En 4D, un BLOB es una serie contigua de bytes de longitud variable, que puede ser tratada como un solo objeto o cuyos bytes pueden ser direccionados individualmente. Un BLOB puede estar vacío (longitud nula) o puede contener hasta 2147483647 bytes (2 GB).

Los BLOBs y la memoria

Un BLOB se carga en su totalidad en memoria. Una variable o array de tipo BLOB se mantiene y existe únicamente en memoria. Un campo de tipo BLOB se carga en memoria desde el disco, como el resto del registro al cual pertenece.
Al igual que otro tipo de campos que pueden retener gran cantidad de datos (campos de tipo Imagen), los campos BLOB no se duplican en memoria cuando usted modifica un registro. Por lo tanto, el resultado devuelto por los comandos **Old** y **Modified** no es significativo cuando se aplica a un campo BLOB.

Visualización de BLOBs

Un BLOB puede contener todo tipo de datos, por lo tanto no tiene una representación en pantalla por defecto. Si usted visualiza un campo o una variable de tipo BLOB en un formulario, siempre aparecerá vacío, sin importar su contenido.

Campos tipo BLOB

Puede utilizar campos de tipo BLOB para guardar todo tipo de datos, hasta 2 GB. No es posible indexar un campo BLOB, por lo tanto debe utilizar una fórmula para buscar registros en valores almacenados en un campo tipo BLOB.

Paso de parámetros, punteros y resultados de funciones

Los BLOBs en 4D pueden pasarse como parámetros a comandos 4D o a rutinas de plug-ins que esperan un parámetro de tipo BLOB. Los BLOBs también pueden pasarse como parámetros a métodos de usuario o ser devueltos como resultado de una función.
Para pasar un BLOB a sus métodos, puede también definir un puntero hacia el BLOB y pasar el puntero como parámetro.
Ejemplos:

```
  ` Declarar una variable de tipo BLOB
C_BLOB(unBlobVar)
  ` El BLOB se pasa como parámetro a un comando 4D
SET BLOB SIZE(unBlobVar;1024*1024)
  ` El BLOB se pasa como parámetro a una rutina externa
$errCode:=Hacer Algo Con Este BLOB(unBlobVar)
  ` El BLOB se pasa como parámetro a un método que devuelve un BLOB
C_BLOB(traerBlob)
traerBlob:=Llenar_Blob(unBlobVar)
  ` Un puntero al BLOB se pasa como parámetro a un método de usuario
CALCULAR BLOB(->unBlobVar)
```

Nota para los desarrolladores de plug-ins 4D: un parámetro de tipo BLOB se declara como "&O" (la letra "O", no el número "0").

Asignación

Puede asignar BLOBs a otros BLOBs.
Ejemplo:

```
  ` Declarar dos variables de tipo BLOB
C_BLOB(vBlobA;vBlobB)
  ` Definir el tamaño del primer BLOB en 10K
SET BLOB SIZE(vBlobA;10*1024)
```

```
  ` Asignar el primer BLOB al segundo
vBlobB:=vBlobA
```

Sin embargo, no se puede aplicar un operador a los BLOBs; no existe una expresión de tipo BLOB.

Direccionar el contenido de un BLOB

Cada byte de un BLOB puede ser direccionado individualmente utilizando llaves *{...}*. En un BLOB, los bytes se numeran de *0* a **N-1**, donde **N** es el tamaño del BLOB. Ejemplo:

```
  ` Declarar una variable de tipo BLOB
C_BLOB(vBlob)
  ` Fijar el tamaño del BLOB en 256 bytes
SET BLOB SIZE(vBlob;256)
  ` El bucle siguiente inicializa los 256 bytes del BLOB en cero
For(vByte;0;BLOB size(vBlob)-1)
   vBlob{vByte}:=0
End for
```

Como es posible direccionar individualmente todos los bytes de un BLOB, usted puede literalmente almacenar todo lo que quiera en un campo o variable tipo BLOB.

Comandos 4D para manejar BLOBs

4D ofrece los siguientes comandos para trabajar con BLOBS:

- **SET BLOB SIZE** redimensiona un campo o variable de tipo BLOB.
- **BLOB size** devuelve el tamaño de un BLOB.
- **DOCUMENT TO BLOB** y **BLOB TO DOCUMENT** le permiten cargar y escribir un documento entero en un campo o una variable de tipo BLOB y al contrario (opcionalmente, confluencia de datos (data fork) y recursos (resources fork) en Macintosh).
- **VARIABLE TO BLOB** y **BLOB TO VARIABLE** como también **SET BLOB SIZE** y **SET BLOB SIZE** le permiten guardar y cargar las variables 4D en BLOBs.
- **COMPRESS BLOB**, **EXPAND BLOB** y **SET BLOB SIZE** le permiten trabajar con BLOBs comprimidos.
- Los comandos **BLOB to integer**, **BLOB to longint**, **BLOB to real**, **BLOB to text**, **INTEGER TO BLOB**, **LONGINT TO BLOB**, **REAL TO BLOB** y **TEXT TO BLOB** le permiten manipular datos estructurados que vienen del disco, de los recursos, del SO, etc.
- **DELETE FROM BLOB**, **INSERT IN BLOB** y **COPY BLOB** permiten manejar rápidamente grandes pedazos de datos dentro de BLOBs.
- **ENCRYPT BLOB** y **DECRYPT BLOB** le permiten encriptar y desencriptar datos en una base 4D.

Estos comandos se describen en este capítulo.
Adicionalmente:

- **C_BLOB** declara una variable de tipo BLOB. Consulte el capítulo **Compilador** para mayor información.
- **ARRAY BLOB** crea o redimensiona un array de tipo BLOB (consulte la sección **Arrays**).
- **GET PASTEBOARD DATA** y **APPEND DATA TO PASTEBOARD** le permiten manejar todo tipo de datos almacenados en el portapapeles. Consulte el capítulo **Gestión de portapapeles** para mayor información.
- **GET RESOURCE** y **_o_SET RESOURCE** le permiten trabajar con todo tipo de recursos almacenados en disco. Consulte el capítulo **Recursos** para mayor información.
- **WEB SEND BLOB** le permite enviar todo tipo de datos a un navegador Web. Consulte el capítulo **Servidor Web** para mayor información.
- **PICTURE TO BLOB**, **BLOB TO PICTURE** y **_o_PICTURE TO GIF** le permiten abrir y convertir imágenes. Para mayor información consulte el tema **Imágenes**.
- **GENERATE ENCRYPTION KEYPAIR** y **GENERATE CERTIFICATE REQUEST** son comandos de encriptación utilizados por el protocolo de conexión segura SSL (Secured Socket Layer). Para mayor información consulte el capítulo **Utilizar el protocolo TLS (HTTPS)**.

BLOB PROPERTIES

BLOB PROPERTIES (BLOB ; comprimido {; descompTam {; tamañoActual}})

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB del cual obtener información
comprimido	Entero largo	⬅	0 = BLOB no está comprimido 1 = BLOB comprimido modo compacto 2 = BLOB comprimido modo rápido
descompTam	Entero largo	⬅	Tamaño del BLOB (en bytes) cuando no está comprimido
tamañoActual	Entero largo	⬅	Tamaño actual del BLOB (en bytes)

Descripción

El comando **BLOB PROPERTIES** devuelve información sobre el BLOB *blob*.
El parámetro *comprimido* devuelve un valor indicando si el BLOB está comprimido. Puede comparar este valor con las siguientes constantes, ubicadas en el tema **BLOB** :

Constante	Tipo	Valor	Comentario
Compact compression mode	Entero largo	1	Compresión interna más compacta (en detrimento de la velocidad a la cual la compresión y descompresión se efectúan). Método por defecto.
Fast compression mode	Entero largo	2	Compresión más rápida en detrimento (y será descomprimido lo más rápido posible), en detrimento de la tasa de compresión (una vez comprimido, el BLOB será más grande).
GZIP best compression mode	Entero largo	-1	Compresión GZIP más compacta (en detrimento de la velocidad a la cual la compresión y decompresión se efectúan).
GZIP fast compression mode	Entero largo	-2	Compresión/descompresión GZIP más rápida (en detrimento de la tasa de compresión).
Is not compressed	Entero largo	0	Sin compresión

Cualquiera que sea el estado de compresión del BLOB, el parámetro *descompTam* devuelve el tamaño del BLOB cuando no está comprimido.
El parámetro *tamañoActual* devuelve el tamaño actual del BLOB. Si el BLOB está comprimido, *tamañoActual* será menor que *descompTam*. Si el BLOB no está comprimido, *tamañoActual* será igual a *descompTam*.

Ejemplo 1

Vea los ejemplos de los comandos **COMPRESS BLOB** y **EXPAND BLOB**.

Ejemplo 2

Después de que un BLOB ha sido comprimido, el siguiente método de proyecto obtiene el porcentaje de espacio ahorrado por la compresión:

```
` Método de proyecto Espacio ahorrado por compresión
` Espacio ahorrado por la compresión (Puntero {; Puntero } ) -> Entero Largo
` Espacio ahorrado por la compresión ( -> BLOB {; -> bytesAhorrados } ) -> Porcentaje

C_POINTER($1;$2)
C_LONGINT($0;$vlComprimido;$vlDescompTam;$vlTamañoActual)

BLOB PROPERTIES($1->;$vlComprimido;$vlDescompTam;$vlTamañoActual)
If($vlDescompTam=0)
   $0:=0
   If(Count parameters>=2)
      $2->:=0
   End if
Else
   $0:=100-(($vlTamañoActual/$vlDescompTam)*100)
   If(Count parameters>=2)
      $2->:=$vlDescompTam-$vlTamañoActual
   End if
End if
```

Después de añadir este método a su aplicación, lo puede utilizar de esta manera:

```
` ...
COMPRESS BLOB(vxBlob)
$vlPorcentaje:=Espacio ahorrado por compresión(->vxBlob;->vlTamañoBlob)
```

```
ALERT("La compresión ahorró "+String(vlBlobSize)+" bytes, "+String($vlPorcentaje;"#0%")+
" de espacio.")
```

BLOB size

BLOB size (BLOB) -> Resultado

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	Campo o variable de tipo BLOB
Resultado	Entero largo	⮌	Tamaño en bytes del BLOB

Descripción

BLOB size devuelve el tamaño del *blob* expresado en bytes.

Ejemplo

La línea de código añade 100 bytes al BLOB *miBlob*:

```
SET BLOB SIZE(miBlob;BLOB size(miBlob)+100)
```

BLOB TO DOCUMENT

BLOB TO DOCUMENT (documento ; BLOB {; *})

Parámetro	Tipo		Descripción
documento	Cadena	➡	Nombre del documento
BLOB	BLOB	➡	Nuevo contenido del documento
*	Operador	➡	*** Obsoleto, no utilizar ***

Descripción

BLOB TO DOCUMENT escribe los datos de *documento* utilizando los datos almacenados en *blob*. Puede pasar el nombre de un documento existente en *documento*. Si el documento no existe, el comando lo crea. Si pasa el nombre de un documento existente, asegúrese de que el documento no esté abierto, de lo contrario se generará un error. Si quiere permitir que el usuario elija el documento, utilice los comandos **Open document** o **Create document** y utilice la variable sistema *documento* (ver ejemplo).

Nota de compatibilidad: el parámetro opcional * (gestión del resource fork en versiones anteriores de Mac OS) ya no se soporta en 4D a partir de 4D v16. Para más información, consulte el manual **Funcionalidades obsoletas y eliminadas**.

Ejemplo

Usted escribe un sistema de información que le permite guardar y buscar rápidamente documentos. En un formulario de entrada de datos, usted crea un botón que le permite guardar un documento que contiene datos cargados previamente en un campo BLOB. El método para este botón puede ser el siguiente:

```
$vhDocRef:=Create document("") ` Guardar el documento de su elección
If(OK=1) ` Si un documento ha sido creado
   CLOSE DOCUMENT($vhDocRef) ` No necesitamos mantenerlo abierto
   BLOB TO DOCUMENT(Document;[SuTabla]SuCampoBLOB) ` Escribir el contenido del documento
   If(OK=0)
 ` Gestionar error
   End if
End if
```

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si el documento está escrito correctamente, de lo contrario toma el valor 0 y se genera un error.

Manejo de errores

- Si trata de reescribir un documento que no existe o que ha sido abierto por otro proceso o aplicación, se genera un error File Manager.
- El espacio del disco puede ser insuficiente para escribir los nuevos contenidos del documento.
- Los errores E/S pueden ocurrir mientras escribe el documento.

En todos los casos, puede interceptar el error utilizando un método de interrupción **ON ERR CALL**.

BLOB to integer

BLOB to integer (BLOB ; byteOrden {; offset}) -> Resultado

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB del cual obtener el valor entero
byteOrden	Entero largo	➡	0 Orden de bytes nativo 1 Orden de bytes Macintosh 2 Orden de bytes PC
offset	Variable	➡	Offset en el BLOB (expresado en bytes)
		⬅	Nuevo offset después de la lectura
Resultado	Entero	↩	Valor entero (2 bytes)

Descripción

El comando **BLOB to integer** devuelve un valor entero (2 bytes) leído del BLOB *blob*.

El parámetro *byteOrden* fija el orden de los bytes ("byte ordering) del valor entero (2 bytes) a leer. Se pasa una de las siguientes constantes predefinidas por 4D:

Constante	Tipo	Valor
Native byte ordering	Entero largo	0
Macintosh byte ordering	Entero largo	1
PC byte ordering	Entero largo	2

Nota sobre la independencia de plataforma: si intercambia BLOBs entre las plataformas Macintosh y PC, es su decisión administrar los temas de byte swapping cuando utilice este comando.

Si especifica la variable del parámetro opcional *offset*, el valor entero (2 bytes) se lee desde el offset (a partir de cero) del BLOB. Si no especifica la variable del parámetro opcional *offset*, se leen los dos primeros bytes del BLOB.

Nota: debe pasar un offset (en bytes) entre *0* (cero) y el tamaño del BLOB menos 2. De lo contrario se genera el error -111.

Después de llamar el comando, la variable se incrementa en el número de bytes leido. Puede reutilizar esa misma variable con otro comando de lectura de BLOBs para leer otro valor.

Ejemplo

El siguiente ejemplo lee 20 valores enteros de un BLOB, a partir del offset 0x200:

```
$vlOffset:=0x200
For($viLoop;0;19)
   $viValor:=BLOB to integer(vxUnBlob;PC byte ordering;$vlOffset)
  ` Hacer algo con $viValor
End for
```

BLOB to list

BLOB to list (BLOB {; offset}) -> Resultado

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB que contiene una lista jerárquica
offset	Entero largo	➡	Offset en el BLOB (expresado en bytes)
		⬅	Nuevo offset después de la lectura
Resultado	ListRef	↩	Referencia de la lista creada recientemente

Descripción

El comando **BLOB to list** crea una nueva lista jerárquica con los datos almacenados en el BLOB *blob* en el offset de bytes (a partir de cero) especificado por *offset* y devuelve un número de referencia de lista jerárquica para esa nueva lista.

Los datos del BLOB deben ser compatibles con el comando. Generalmente, usted utiliza BLOBs llenados previamente con el comando **LIST TO BLOB**.

Si no especifica el parámetro opcional *offset*, los valores de las lista se leen desde el comienzo del BLOB. Si trabaja con un BLOB en el cual se almacenan muchas variables o listas, debe pasar el parámetro *offset* y, adicionalmente, debe pasar una variable numérica. Antes del llamado, fije esta variable numérica al offset apropiado. Después del llamado, la misma variable numérica devuelve el offset de la variable siguiente almacenada en el BLOB.

Después del llamado, si la lista jerárquica ha sido creada correctamente, la variable OK toma el valor 1. Si la operación no pueder ser efectuada, la variable OK toma el valor 0; por ejemplo, si no hay suficiente memoria.

Nota sobre la independencia de plataforma: BLOB to list y **LIST TO BLOB** utilizan un formato interno 4D para administrar listas almacenadas en BLOBs. La ventaja es que usted no tiene que preocuparse por la conversión de bytes (byte swapping) entre plataformas cuando utilice estos dos comandos. En otras palabras, un BLOB creado en Windows utilizando estos dos comandos puede ser reutilizados en Macintosh y viceversa.

Ejemplo

En este ejemplo, el método de un formulario de entrada extrae una lista de un campo BLOB antes de que el formulario aparezca en la pantalla, y lo almacena nuevamente en el campo BLOB si la entrada de datos se valida:

```
` Método de formulario [Cosas por hacer];"Entrada"

Case of

   :(Form event=On Load)
      hList:=BLOB to list([Cosas por hacer]Ideas)
      If(OK=0)
         hList:=New list
      End if

   :(Form event=On Unload)
      CLEAR LIST(hList;*)

   :(bValidate=1)
      LIST TO BLOB(hList;[Cosas por hacer]Ideas)

End case
```

Variables y conjuntos del sistema

La variable OK toma el valor 1 si la lista se crea correctamente, de lo contrario toma el valor 0.

BLOB to longint

BLOB to longint (BLOB ; byteOrden {; offset}) -> Resultado

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB del cual obtener el valor entero largo
byteOrden	Entero largo	➡	0 Orden de bytes nativo 1 Orden de bytes Macintosh 2 Orden de bytes PC
offset	Variable	➡	Offset en el BLOB (expresado en bytes)
		⬅	Nuevo offset después de la lectura
Resultado	Entero largo	↩	Valor entero largo (4 bytes)

Descripción

El comando **BLOB to longint** devuelve un valor de tipo Entero largo (4 bytes) leído del BLOB *blob*.

El parámetro *byteOrder* fija el orden de los bytes ("byte ordering") del valor Entero largo (4 bytes) a leer. Se pasa una de las siguientes constantes predefinidas de 4D:

Constante	Tipo	Valor
Native byte ordering	Entero largo	0
Macintosh byte ordering	Entero largo	1
PC byte ordering	Entero largo	2

Nota sobre la independencia de plataforma: si intercambia BLOBs entre las plataformas Macintosh y PC, es su decisión administrar los temas de byte swapping cuando utilice este comando.

Si especifica la variable del parámetro opcional *offset*, el entero largo (4 bytes) se lee en el offset (a partir de cero) del BLOB. Si no especifica la variable del parámetro opcional *offset*, se leen los cuatro primeros bytes del BLOB.

Nota: debe pasar un valor de offset entre *0* (cero) y el tamaño del BLOB menos 4. Si no lo hace, se genera el error -111.

Después de llamar el comando, la variable se incrementa en el número de bytes leídos. Por lo tanto, puede reutilizar la misma variable con otro comando de lectura de BLOBs para leer otro valor.

Ejemplo

El siguiente ejemplo lee 20 valores de tipo entero largo de un BLOB, a partir del offset 0x200:

```
$vlOffset:=0x200
For($viLoop;0;19)
   $vlValor:=BLOB to longint(vxUnBlob;PC byte ordering;$vlOffset)
  ` Hacer algo con $vlValor
End for
```

BLOB to real

BLOB to real (BLOB ; formatoReal {; offset}) -> Resultado

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB del cual obtener el valor de tipo Real
formatoReal	Entero largo	➡	0 Formato real nativo 1 Formato real extendido 2 Formato real doble Macintosh 3 Formato real doble Windows
offset	Variable	➡	Offset en el BLOB (expresado en bytes)
		⬅	Nuevo offset después de la lectura
Resultado	Real	↩	Valor real

Descripción

El comando **BLOB to real** devuelve un valor de tipo Real leído del BLOB *blob*.

El parámetro *formatoReal* fija el formato interno y el orden de bytes del valor de tipo Real a leer. Se pasa una de las siguientes constantes predefinidas de 4D:

Constante	Tipo	Valor
Native real format	Entero largo	0
Extended real format	Entero largo	1
Macintosh double real format	Entero largo	2
PC double real format	Entero largo	3

Nota sobre la independencia de plataforma: si intercambia BLOBs entre las plataformas Macintosh y PC, es su decisión administrar los temas de byte swapping cuando utilice este comando.

Si especifica la variable del parámetro opcional *offset*, el valor real es leído desde el offset (a partir de cero) del BLOB. Si no especifica la variable del parámetro opcional *offset*, se leen los primeros 8 o 10 bytes del BLOB.

Nota: debe pasar un valor de offset entre *0* (cero) y el tamaño del BLOB menos 8 o 10. Si no lo hace, se genera un error -111.

Después de llamar el comando, la variable se incrementa en el número de bytes leído. Por lo tanto, puede reutilizar la misma variable con otro comando de lectura de BLOB para leer otro valor.

Ejemplo

El siguiente ejemplo lee 20 valores reales de un BLOB, a partir del offset 0x200:

```
$vlOffset:=0x200
For($viLoop;0;19)
   $vrValor:=BLOB to real(vxSomeBlob;PC byte ordering;$vlOffset)
  ` Hacer algo con $vrValor
End for
```

BLOB to text

BLOB to text (BLOB ; formatoTexto {; offset {; longitudTexto}}) -> Resultado

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB del cual obtener el texto
formatoTexto	Entero largo	➡	Formato y conjunto de caracteres de texto
offset	Variable	➡	Offset en el BLOB (expresado en bytes)
		⬅	Nuevo offset después de la lectura
longitudTexto	Entero largo	➡	Número de caracteres a leer
Resultado	Texto	↩	Valor del texto

Descripción

El comando **BLOB to text** devuelve un valor de tipo Texto leído del BLOB *blob*.

El parámetro *formatoTexto* fija el formato interno y el conjunto de caracteres del valor de tipo Texto a leer. En las bases de datos creadas a partir de la versión 11, 4D utiliza por defecto el conjunto de caracteres Unicode (UTF8) para la gestión de textos. Por compatibilidad, este comando permite "forzar" la utilización del conjunto de caracteres Mac Roman (conjunto de caracteres utilizado en las versiones anteriores de 4D). La elección del conjunto de caracteres se efectúa vía el parámetro *formatoTexto.* Para hacer eso, pase una de las siguientes constantes (del tema **BLOB**) en el parámetro *formatoTexto*:

Constante	Tipo	Valor
Mac C string	Entero largo	0
Mac Pascal string	Entero largo	1
Mac text with length	Entero largo	2
Mac text without length	Entero largo	3
UTF8 C string	Entero largo	4
UTF8 text with length	Entero largo	5
UTF8 text without length	Entero largo	6

Notas:

- Las constantes "UTF8" sólo pueden ser utilizadas cuando la aplicación se ejecuta en modo Unicode.
- La constante **Mac Text with length** no puede trabajar con textos de más de 32 KB.
- Si quiere trabajar con conjuntos de caracteres diferentes de UTF8, utilice el comando Convert to text.

Para mayor información sobre estas constantes y los formatos que representan, consulte la descripción del comando TEXT TO BLOB.

Atención: el número de caracteres a leer se determina por el parámetro *formatoTexto*, EXCEPTO para el formato **Text without length** y **UTF8 Text without length**, para el cual usted DEBE especificar el número de caracteres a leer en el parámetro *longitudTexto*. Para los otros formatos, *longitudTexto* se ignora y usted puede omitirlo.

Si especifica la variable del parámetro opcional *offset*, el valor del texto se lee en el offset (a partir de cero) del BLOB. Si no especifica la variable del parámetro opcional *offset*, se leen los primeros bytes del BLOB de acuerdo al valor pasado en *textFormato*. Note que debe pasar la variable del parámetro *offset* cuando esté leyendo texto sin longitud.

Nota: debe pasar un valor de offset entre *0* (cero) y el tamaño del BLOB menos el tamaño del texto a leer. Si no lo hace, el resultado de la función es impredecible.

Después de la ejecución del comando, la variable se incrementa en el número de bytes leídos. Por lo tanto, puede reutilizar la misma variable con otro comando de lectura de BLOBs para leer otro valor.

BLOB TO VARIABLE

BLOB TO VARIABLE (BLOB ; variable {; offset})

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB que contiene variables 4D
variable	Variable	⬅	Variable a escribir con el contenido del BLOB
offset	Entero largo	➡	Posición de la variable en el BLOB
		⬅	Posición de la variable siguiente en el BLOB

Descripción

El comando **BLOB TO VARIABLE** reescribe la variable *variable* con los datos almacenados en el BLOB *blob* en el offset de bytes (a partir de cero) especificado por *offset*.

Los datos en el BLOB deben ser compatibles con la variable de destino. Generalmente, usted utilizará los BLOBs que usted ha llenado previamente utilizando el comando **VARIABLE TO BLOB**.

Si no especifica el parámetro opcional *offset*, los datos de la variable se leen desde el inicio del BLOB. Si usted está trabajando con un BLOB que contiene muchas variables, debe pasar el parámetro *offset* y, adicionalmente, debe pasar una variable numérica. Antes de llamar el comando, defina esta variable numérica en el offset correspondiente. Después de llamar el comando, la misma variable númerica devuelve el offset de la siguiente variable almacenada en el BLOB.

Nota: BLOB TO VARIABLE soporta las variables objeto y colección de tipo **C_OBJECT** y **C_COLLECTION**. Para mayor información, consulte el comando **VARIABLE TO BLOB**.

Después de llamar el comando, si la variable ha sido reescrita con éxito, la variable OK toma el valor 1. Si la operación no se pudo realizar, la variable OK toma el valor 0; por ejemplo, si no hay suficiente memoria.
Nota sobre la independencia de plataforma: BLOB TO VARIABLE y **VARIABLE TO BLOB** utilizan un formato interno de 4D para administrar las variables almacenadas en los BLOBs. La ventaja es que usted no tiene que preocuparse por la conversión de bytes (byte swapping) entre plataformas mientras utiliza estos dos comandos. En otras palabras, un BLOB creado en Windows utilizando cualquiera de estos dos comandos puede ser reutilizado en Macintosh y viceversa.

Ejemplo

Vea los ejemplos para el comando **VARIABLE TO BLOB**.

Variables y conjuntos del sistema

La variable OK toma el valor 1 si la variable ha sido reescrita correctamente, de lo contrario toma el valor 0.

COMPRESS BLOB

COMPRESS BLOB (BLOB {; compresion})

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB a comprimir
compresion	Entero largo	➡	Si no se omite: 1, compresión máxima posible 2, velocidad de compresión máxima

Descripción

El comando **COMPRESS BLOB** comprime el BLOB *blob* utilizando el algoritmo de compresión interno de 4D.

El parámetro opcional *compresion* le permite definir la forma en que la que el BLOB se comprimirá. Pase en este parámetro una de las siguientes constantes, ubicadas en el tema **BLOB** :

Constante	Tipo	Valor	Comentario
Compact compression mode	Entero largo	1	Compresión interna más compacta (en detrimento de la velocidad a la cual la compresión y descompresión se efectúan). Método por defecto.
Fast compression mode	Entero largo	2	Compresión más rápida en detrimento (y será descomprimido lo más rápido posible), en detrimento de la tasa de compresión (una vez comprimido, el BLOB será más grande).
GZIP best compression mode	Entero largo	-1	Compresión GZIP más compacta (en detrimento de la velocidad a la cual la compresión y decompresión se efectúan).
GZIP fast compression mode	Entero largo	-2	Compresión/descompresión GZIP más rápida (en detrimento de la tasa de compresión).

Si pasa otro valor o si omite el parámetro *compresion*, se utiliza el método de compresión 1 (algoritmo interno compacto).

Después de llamar este comando, la variable OK toma el valor 1 si el BLOB fue comprimido correctamente. Si la compresión no se pudo realizar, la variable OK toma el valor 0. Si la compresión no se pudo efectuar por falta de memoria o porque el tamaño actual del blob es menor de 255 bytes, no se genera un error, y el método continua su ejecución.

En otros casos si el error es causado por un problema más importante (el BLOB está dañado), se genera el error -10600. Este error, puede ser interceptado con la ayuda de un método instalado por el comando **ON ERR CALL**.

Después de comprimir un BLOB, puede expandirlo utilizando el comando **EXPAND BLOB**.

Para detectar si un BLOB ha sido comprimido, utilice el comando **BLOB PROPERTIES.**

Advertencia: un BLOB comprimido continua siendo un BLOB, de manera que no hay nada que le impida modificar su contenido. Sin embargo, si lo modifica, el comando **EXPAND BLOB** no podrá descomprimir el BLOB correctamente.

Ejemplo 1

Este ejemplo prueba si el BLOB *vxMiBlob* está comprimido, y si no lo está, lo comprime:

```
BLOB PROPERTIES(</Gen9><span class="rte4d_prm">vxMiBlob</span><Gen9>;$vlComprimido;$vlTamañoExpandido;$vlTamañoActual)
If($vlComprimido=Is not compressed)
   COMPRESS BLOB(vxMiBlob)
End if
```

Sin embargo observe, que si aplica **COMPRESS BLOB** a un BLOB que ya ha sido comprimido, el comando lo detecta y no hace nada.

Ejemplo 2

Este ejemplo le permite seleccionar un documento y luego comprimirlo:

```
$vhDocRef :=Open document("")
If(OK=1)
   CLOSE DOCUMENT($vhDocRef)
   DOCUMENT TO BLOB(Document;vxBlob)
   If(OK=1)
      COMPRESS BLOB(vxBlob)
      If(OK=1)
         BLOB TO DOCUMENT(Document;vxBlob)
      End if
   End if
End if
```

Ejemplo 3

Envío de datos HTTP brutos comprimidos en GZIP:

```
COMPRESS BLOB($blob;GZIP Best compression mode)
C_TEXT($vEncoding)
$vEncoding:="Content-encoding: gzip"
WEB SET HTTP HEADER($vEncoding)
WEB SEND RAW DATA($blob ;*)
```

Variables y conjuntos del sistema

La variable OK toma el valor 1 si el BLOB se comprime correctamente; de lo contrario, toma el valor 0.

COPY BLOB

COPY BLOB (srcBLOB ; dstBLOB ; srcOffset ; dstOffset ; numero)

Parámetro	Tipo		Descripción
srcBLOB	BLOB	➡	BLOB fuente
dstBLOB	BLOB	➡	BLOB de destino
srcOffset	Entero largo	➡	Posición de la fuente para la copia
dstOffset	Entero largo	➡	Posición de destino para la copia
numero	Entero largo	➡	Número de bytes a copiar

Descripción

El comando **COPY BLOB** copia el número de bytes especificado por *nombre* del BLOB *srcBLOB* al BLOB *dstBLOB*.

La copia comienza en la posición (expresada con relación al comienzo del BLOB fuente) definida por *srcOffset* y toma lugar a partir de la posición (expresada con relación al comienzo del BLOB de destino) especificada por *dstOffset*.

Nota: el BLOB de destino puede redimensionarse si es necesario.

DECRYPT BLOB

DECRYPT BLOB (aDescifrar ; enviarLlavePub {; recepLlavePriv})

Parámetro	Tipo		Descripción
aDescifrar	BLOB	⇒	Datos a descifrar
		⇐	Datos descifrados
enviarLlavePub	BLOB	⇒	Llave pública del emisor
recepLlavePriv	BLOB	⇒	Llave privada del receptor

Descripción

El comando **DECRYPT BLOB** descifra el contenido del BLOB a**Descrifrar** utilizando la llave pública del emisor *enviarLlavePub* y opcionalmente la llave privada del receptor *recepLlavePriv*.

El BLOB que contiene la llave pública del emisor se pasa en el parámetro *enviarLlavePub*. Esta llave ha sido generada por el emisor utilizando el comando **GENERATE ENCRYPTION KEYPAIR** y tiene que ser enviada al receptor.

El BLOB que contiene la llave privada del receptor puede pasarse en el parámetro opcional *recepLlavePriv*. En este caso, el receptor ha generado un par de llaves de cifrado con el comando **GENERATE ENCRYPTION KEYPAIR** y tiene que enviar su llave pública al emisor. El sistema de cifrado a dos llaves garantiza que el mensaje ha sido cifrado sólo por el emisor y que puede ser decodificado únicamente por el receptor. Para mayor información sobre el sistema de codificación de dos llaves, consulte la rutina **ENCRYPT BLOB**.

El comando **DECRYPT BLOB** ofrece una funcionalidad de verificación de integridad (checksum) con el fin de evitar toda modificación del contenido del BLOB (deliberada o no). Si el BLOB cifrado está dañado o modificado, el comando no hará nada y devolverá un error.

Ejemplo

Consulte los ejemplos dados por el comando **ENCRYPT BLOB**.

DELETE FROM BLOB

DELETE FROM BLOB (BLOB ; offset ; numero)

Parámetro	Tipo		Descripción
BLOB	BLOB	⇒	BLOB del cual borrar los bytes
offset	Entero largo	⇒	Offset a partir del cual borrar los bytes
numero	Entero largo	⇒	Número de bytes a borrar

Descripción

El comando **DELETE FROM BLOB** borra el número de bytes especificado por *numero* del BLOB *blob* a partir de la posición especificada por *offset* (expresada de manera relativa al comienzo del BLOB). El tamaño del BLOB se reduce en *numero* de bytes.

DOCUMENT TO BLOB

DOCUMENT TO BLOB (documento ; BLOB {; *})

Parámetro	Tipo		Descripción
documento	Cadena	➡	Nombre del documento
BLOB	BLOB	➡	Campo o variable de tipo BLOB a recibir el documento
		⬅	Contenido del documento
*	Operador	➡	*** Obsoleto, no utilizar ***

Descripción

DOCUMENT TO BLOB carga el contenido de *documento* en *blob*. Debe pasar el nombre de un documento existente que no esté abierto, de lo contrario se generará un error. Para dejar que el usuario seleccione el documento a cargar en el BLOB, utilice el comando **Open document** y la variable sistema *document* (ver ejemplo).

Nota de compatibilidad: El parámetro opcional * (gestión del *resource fork* en versiones anteriores de Mac OS) ya no se admite en 4D a partir de 4D v16. Para más información, consulte el manual **Funcionalidades obsoletas y eliminadas**.

Ejemplo

Usted escribe un Sistema de información que le permite guardar y buscar rápidamente documentos. En un formulario de entrada de datos, usted crea un botón que le permite cargar un documento en un campo tipo BLOB. El método para este botón puede ser:

```
$vhDocRef:=Open document("") ` Seleccionar un documento
If(OK=1) ` Si un documento ha sido seleccionado
   CLOSE DOCUMENT($vhDocRef) ` No necesitamos mantenerlo abierto
   DOCUMENT TO BLOB(Document;[SuTabla]SuCampoBLOB) ` Cargar el documento
   If(OK=0)
 ` Manejar error
   End if
End if
```

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si el documento se carga correctamente, de lo contrario OK toma el valor 0 y se genera un error.

Manejo de errores

- Si trata de cargar (en un BLOB) un documento que no existe o que ya ha sido abierto por otro proceso o aplicación, se genera un error File Manager.
- Un error de E/S puede ocurrir si el documento está bloqueado, si está ubicado en un volumen bloqueado, o si hay un problema en la lectura del documento.
- Si no hay suficiente memoria para cargar el documento, se genera un error -108.

En todos los casos, puede interceptar el error utilizando un método de interrupción **ON ERR CALL**.

ENCRYPT BLOB

ENCRYPT BLOB (aCifrar ; enviarLlavePriv {; recepLlavePub})

Parámetro	Tipo		Descripción
aCifrar	BLOB	➡	Datos a cifrar
		⬅	Datos cifrados
enviarLlavePriv	BLOB	➡	Llave privada del emisor
recepLlavePub	BLOB	➡	Llave pública del receptor

Descripción

El comando ENCRYPT BLOB cifra el contenido del BLOB *aCifrar* con la ayuda de la llave privada del emisor *enviarLlavePriv*, así como también opcionalmente la llave pública del receptor *recepLlavePub*. Estas llaves deben ser generadas por el comando **GENERATE ENCRYPTION KEYPAIR** (en el tema "Protocolo de seguridad").

Nota: este comando utiliza el algoritmo y las funcionalidades de cifrado del protocolo TLS. Para utilizar este comando, asegúrese de que los componentes necesarios para el funcionamiento del protocolo TLS estén instalados correctamente en su equipo, incluso si no quiere utilizar TLS para conexiones con el servidor Web 4D. Para información detallada sobre este protocolo, consulte la sección **Utilizar el protocolo TLS (HTTPS)**.

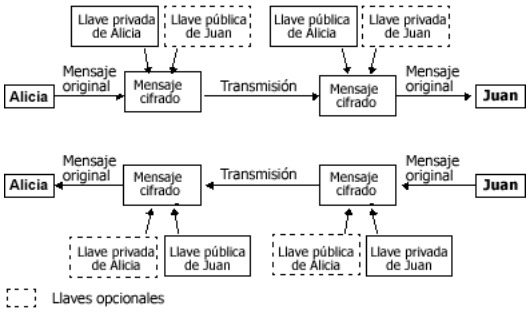
- Si una llave se utiliza para el cifrado (la llave privada del emisor), sólo las personas que tengan la llave pública podrán leer la información. Este sistema garantiza que el emisor haya cifrado la información.
- El uso simultáneo de la llave privada del emisor y la llave pública del receptor garantiza que sólo un receptor podrá leer la información.

El BLOB que contiene las llaves tiene un formato interno PKCS. Este formato estándar, multiplataforma, permite intercambiar o manipular las llaves simplemente haciendo copiar-pegar en un email o archivo de texto.

Una vez ejecutado el comando, el BLOB *aCifrar* contiene los datos cifrados que serán descifrados sólo con el comando **DECRYPT BLOB**, con la llave pública del emisor pasada como parámetro.

Además, si la llave pública opcional ha sido utilizada para cifrar la información, la llave privada del receptor también será necesaria para descifrar.

Principio de cifrado con llaves públicas y privadas para el intercambio de mensajes entre dos personas, "Alicia" y "Juan":



Nota: el algoritmo de cifrado contiene una función de verificación de integridad (checksum), con el fin de evitar toda modificación del contenido del BLOB (intencional o no). Por lo tanto, un BLOB cifrado no debe modificarse de lo contrario no podría descifrarse.

Optimización de comandos de cifrado

El cifrado de los datos disminuye la velocidad de ejecución de sus aplicaciones, especialmente si se utiliza un par de llaves. Sin embargo, puede considerar los siguientes consejos de optimización:

- Dependiendo de la memoria disponible, el comando se ejecutará en modo "sincrónico" o "asincrónico".
  El modo asincrónico es más rápido, ya que no bloquea los otros procesos. Este modo se utiliza automáticamente si la memoria disponible es por lo menos dos veces del tamaño de los datos a cifrar.

De lo contrario, por razones de seguridad se utiliza el modo sincrónico. Este modo es menos rápido ya que bloquea los otros procesos.

- En el caso de BLOBs grandes, puede cifrar sólo una pequeña parte "estratégica" del BLOB con el fin de reducir el tamaño de los datos a procesar como también el tiempo de ejecución.

Ejemplo

- **Utilización de una sola llave**

Una empresa quiere garantizar la confidencialidad de los datos almacenados en una base 4D. La empresa debe enviar esta información con regularidad a sus filiales a través de archivos, vía Internet.

1) La empresa genera un par de llaves con el comando **GENERATE ENCRYPTION KEYPAIR**:

```
`Método GENERAR_LLAVES_TXT
C_BLOB($BLlavePublica;$BLlavePrivada)
GENERATE ENCRYPTION KEYPAIR($BLlavePrivada;$BLlavePublica)
```

```
BLOB TO DOCUMENT("LlavePublica.txt";$BLlavePublica)
BLOB TO DOCUMENT("LlavePrivada.txt";$BLlavePrivada)
```

2) La empresa conserva la llave privada, y envía a cada filial una copia del documento que contiene la llave pública. Para máxima seguridad, la llave debe copiarse en un disco y ser enviada físicamente a las subsidiarias.

3) Luego la empresa copia la información confidencial, (almacenada por ejemplo en el campo tipo texto) en BLOBs y los cifra con la llave privada:

```
  `Método CIFRAR_INFO
C_BLOB($vbCifrado;$vbLlavePrivada)
C_TEXT($vtCifrado)

$vtCifrado:=[Privado]Info
VARIABLE TO BLOB($vtCifrado;$vbCifrado)
DOCUMENT TO BLOB("LlavePrivada.txt";$vbLlavePrivada)
If(OK=1)
   ENCRYPT BLOB($vbCifrado;$vbLlavePrivada)
   BLOB TO DOCUMENT("Update.txt";$vbCifrado)
End if
```

4) Los archivos actualizados pueden enviarse a las filiales (a través de un canal no seguro como Internet). Si una tercera persona intercepta el archivo cifrado, no podrá descifrarlo sin la llave pública.

5) Cada filial puede descifrar el documento con la llave pública:

```
  `Método DESCIFRAR_INFO
C_BLOB($vbCifrado;$vbLlavePublica)
C_TEXT($vtDescifrado)
C_TIME($vtDocRef)

ALERT("Por favor seleccione un documento cifrado.")
$vtDocRef:=Open document("")  `Seleccione Update.txt
If(OK=1)
   CLOSE DOCUMENT($vtDocRef)
   DOCUMENT TO BLOB(Document;$vbCifrado)
  `Su llave privada es cargada
   DOCUMENT TO BLOB("LlavePublica.txt";$vbLlavePublica)
   If(OK=1)
      DECRYPT BLOB($vbCifrado;$vbLlavePublica)
      BLOB TO VARIABLE($vbCifrado;$vtDescifrado)
      CREATE RECORD([Privado])
      [Privado]Info:=$vtDescifrado
      SAVE RECORD([Privado])
   End if
End if
```

- **Utilización de llaves pares**

Una empresa quiere utilizar el Internet para intercambiar información. Cada filial recibe información privada y también envía información a la oficina principal. Por lo tanto hay dos requerimientos:
- Sólo el destinatario debe poder leer el mensaje,
- El destinatario debe tener prueba de que el mensaje fue enviado por el emisor.

1) La oficina principal y cada filial genera sus propios pares de llaves (con el método **GENERAR_LLAVES_TXT**).

2) La llave privada se mantiene en secreto para ambas partes. Cada filial envía su llave pública a la oficina principal, quien a su vez envía su llave pública. Esta transferencia de llaves no necesariamente tiene que ser efectuada por un canal protegido ya que la llave pública no es suficiente para descifrar el mensaje.

3) Para cifrar la información a enviar, la filial o la oficina principal ejecuta el método **CIFRAR_INFO_2** el cual utiliza la llave privada del emisor y la llave pública del destinatario para cifrar la información:

```
  `Método </Gen9><span class="rte4d_cmd">CIFRAR_INFO_2</span><Gen9>
C_BLOB($vbCifrado;$vbLlavePrivada;$vbLlavePublica)
C_TEXT($vtCifrado)
C_TIME($vtDocRef)

$vtEncrypt:=[Privado]Info
VARIABLE TO BLOB($vtCifrado;$vbCifrado)
  ` Su propia llave privada se carga...
DOCUMENT TO BLOB("LlavePrivada.txt";$vbLlavePrivada)
If(OK=1)
  ` ...y la llave pública del destinatario
   ALERT("Por favor seleccione la llave pública del destinatario.")
   $vhDocRef:=Open document("")  `Llave pública a cargar
   If(OK=1)
```

```
        CLOSE DOCUMENT($vtDocRef)
        DOCUMENT TO BLOB(Document;$vbLlavePublica)
    `Cifrado del BLOB con dos llaves como parámetros
        ENCRYPT BLOB($vbCifrado;$vbLlavePrivada;$vbLlavePublica)
        BLOB TO DOCUMENT("Update.txt";$vbCifrado)
    End if
End if
```

4) El archivo cifrado puede entonces ser enviado al destinatario vía Internet. Si una tercera persona lo intercepta, no podrá descifrar el mensaje, incluso si tiene las llaves públicas porque no tendrá la llave privada del destinatario.

5) Cada destinatario puede descifrar el documento utilizando su propia llave privada y la llave pública del emisor:

```
    `Método </Gen9><span class="rte4d_cmd">CIFRAR_INFO_2</span><Gen9>
C_BLOB($vbCifrado;$vbLlavePrivada;$vbLlavePublica)
C_TEXT($vtCifrado)
C_TIME($vtDocRef)

ALERT("Por favor seleccione la llave pública del destinatario.")
$vhDocRef:=Open document("")  `Llave pública a cargar
If(OK=1)
    CLOSE DOCUMENT($vhDocRef)
    DOCUMENT TO BLOB(Document;$vbCifrado)
  `Se carga su propia llave privada
    DOCUMENT TO BLOB("LlavePrivada.txt";$vbLlavePrivada)
    If(OK=1)
  ` ...y la llave pública del emisor
        ALERT("Por favor seleccione la llave pública del emisor.")
        $vhDocRef:=Open document("")  `Llave pública a cargar
        If(OK=1)
            CLOSE DOCUMENT($vhDocRef)
            DOCUMENT TO BLOB(Document;$vbLlavePublica)
  `Descifrar el BLOB con dos llaves como parámetros
            DECRYPT BLOB($vbCifrado;$vbLlavePublica;$vbLlavePrivada)
            BLOB TO VARIABLE($vbCifrado;$vtDescifrado)
            CREATE RECORD([Privado])
            [Privado]Info:=$vtDescifrado
            SAVE RECORD([Privado])
        End if
    End if
End if
```

EXPAND BLOB

EXPAND BLOB (BLOB)

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB a descomprimir

Descripción

El comando **EXPAND BLOB** descomprime el BLOB *blob* que fue previamente comprimido utilizando el comando **COMPRESS BLOB.**

Después de llamar el comando, la variable OK toma el valor 1 si el BLOB ha sido descomprimido. Si no pudo realizar la expansión, la variable OK toma el valor 0.
Si la expansión no se pudo realizar por falta de memoria, no se genera un error y el método reasume su ejecución.
En otros casos, (el BLOB no ha sido comprimido o está dañado), se genera el error -10600. Este error puede interceptarse utilizando el comando **ON ERR CALL**.
Para verificar si un BLOB ha sido comprimido, utilice el comando **BLOB PROPERTIES**.

Ejemplo 1

Este ejemplo prueba si el BLOB *vxMiBlob* está comprimido, si es así lo descomprime:

```
BLOB PROPERTIES(vxMiBlob;$vlComprimido;$vlTamañoExpandido;$vlTamañoActual)
If($vlComprimido#Is not compressed)
   EXPAND BLOB(vxMiBlob)
End if
```

Ejemplo 2

Este ejemplo le permite seleccionar un documento y descomprimirlo, si está comprimido:

```
$vhDocRef :=Open document("")
If(OK=1)
   CLOSE DOCUMENT($vhDocRef)
   DOCUMENT TO BLOB(Document;vxBlob)
   If(OK=1)
      BLOB PROPERTIES(vxBlob;$vlComprimido;$vlTamañoExpandido;$vlTamañoActual)
      If($vlComprimido#Is not compressed)
         EXPAND BLOB(vxBlob)
         If(OK=1)
            BLOB TO DOCUMENT(Document;vxBlob)
         End if
      End if
   End if
End if
```

Variables y conjuntos del sistema

La variable OK toma el valor 1 si el BLOB ha sido descomprimido con éxito, de lo contrario toma el valor 0.

INSERT IN BLOB

INSERT IN BLOB (BLOB ; offset ; numero {; relleno})

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB en el cual insertar los bytes
offset	Entero largo	➡	Posición de inicio de inserción de los bytes
numero	Entero largo	➡	Número de bytes a insertar
relleno	Entero largo	➡	Valor de byte por defecto (0x00..0xFF) 0x00 si se omite

Descripción

El comando **INSERT IN BLOB** inserta el número de bytes especificado por *numero* en el BLOB *blob* en la posición especificada por *offset*. El BLOB se vuelve *numero* bytes más grande.

Si no especifica el parámetro opcional *relleno*, los bytes insertados en el BLOB se fijan para *0x00*. De lo contrario, los bytes se definen para el valor pasado en *defecto* (modulo 256 — 0..255).

Antes de llamar el comando, pase en la variable del parámetro *offset* la posición de la inserción relativa al comienzo del BLOB.

INTEGER TO BLOB

INTEGER TO BLOB (entero ; BLOB ; byteOrden {; offset | *})

Parámetro	Tipo		Descripción
entero	Entero largo	➡	Valor entero a escribir en el BLOB
BLOB	BLOB	➡	BLOB a recibir el valor entero
byteOrden	Entero largo	➡	0 Orden de bytes en modo nativo 1 Orden de los bytes Macintosh 2 Orden de los bytes PC
offset | *	Variable, Operador	➡	Offset expresado en bytes en el BLOB o * para añadir el valor
		⬅	Nuevo offset después de la escritura si se omite *

Descripción

El comando **INTEGER TO BLOB** escribe el valor entero (2 bytes) *entero* en el BLOB *blob*.

El parámetro *byteOrden* fija el orden de los bytes ("byte ordering) del valor entero a escribir (2 bytes). Usted pasa uno de las siguientes constantes predefinidas suministradas 4D:

Constante	Tipo	Valor
Native byte ordering	Entero largo	0
Macintosh byte ordering	Entero largo	1
PC byte ordering	Entero largo	2

Nota sobre la independencia de plataforma: si intercambia BLOBs entre las plataformas Macintosh y PC, es su decisión administrar los temas de byte swapping cuando utilice este comando.

Si especifica el parámetro opcional *, el valor entero de 2 bytes se añade al BLOB y el tamaño del BLOB se extiende en consecuencia. Utilizando el parámetro opcional *, puede almacenar secuencialmente todos los valores de tipo *Entero*, *Entero largo*, **Real** o *Texto* (ver otros comandos de BLOB) en un BLOB, siempre y cuando tenga memoria disponible.

Si no especifica el parámetro opcional * o la variable del parámetro *offset*, el valor entero de 2 bytes se almacena al comienzo del BLOB, sobreescribiendo su contenido anterior; el tamaño del BLOB se ajusta en consecuencia.

Si pasa la variable del parámetro *offset*, el valor entero de 2 bytes se escribe a partir del offset (comenzando desde cero) del BLOB. Sin importar donde escriba el valor entero de 2 bytes, el tamaño del BLOB aumenta de acuerdo a la ubicación pasada (hasta en 2 bytes, si es necesario). Los bytes recientemente definidos, diferentes a los que está escribiendo, se inicializan en cero.

Después de llamar el comando, se devuelve la variable del parámetro *offset*, incrementada en el número de bytes que se han escrito. Por lo tanto, usted puede reutilizar esa misma variable con otro comando de escritura de BLOB para escribir otro valor.

Ejemplo 1

Después de la ejecución de este código:

```
INTEGER TO BLOB(0x0206;vxBlob;Native byte ordering)
```

- El tamaño de *vxBlob* es 2 bytes
- En Macintosh *vxBLOB{0}* = *$02* y *vxBLOB{1}* = *$06*
- En PC *vxBLOB{0}* = *$06* y *vxBLOB{1}* = *$02*

Ejemplo 2

Después de la ejecución de este código:

```
INTEGER TO BLOB(0x0206;vxBlob;Macintosh byte ordering)
```

- El tamaño de *vxBlob* es 2 bytes
- En todas las plataformas *vxBLOB{0}* = *$02* and *vxBLOB{1}* = *$06*

Ejemplo 3

Después de la ejecución de este código:

```
INTEGER TO BLOB(0x0206;vxBlob;PC byte ordering)
```

- El tamaño de *vxBlob* is 2 bytes
- En todas las plataformas *vxBLOB{0}* = *$06* and *vxBLOB{1}* = *$02*

Ejemplo 4

Después de la ejecución de este código:

```
SET BLOB SIZE(vxBlob;100)
INTEGER TO BLOB(0x0206;vxBlob;PC byte ordering;*)
```

- El tamaño de *vxBlob* es 102 bytes
- En todas las plataformas *vxBLOB{100}* = *$06* and *vxBLOB{101}* = *$02*
- Los otros bytes del BLOB no cambian

Ejemplo 5

Después de la ejecución de este código:

```
SET BLOB SIZE(vxBlob;100)
vlOffset:=50
INTEGER TO BLOB(518;vxBlob;Macintosh byte ordering;vlOffset)
```

- El tamaño de *vxBlob* es 100 bytes
- En todas las plataformas *vxBLOB{50}* = *$02* and *vxBLOB{51}* = *$06*
- Los otros bytes del BLOB no cambian
- La variable *vlOffset* se ha incrementado en 2 (y ahora es igual a 52)

LIST TO BLOB

LIST TO BLOB (lista ; BLOB {; *})

Parámetro	Tipo		Descripción
lista	ListRef	➡	Lista jerárquica a almacenar en el BLOB
BLOB	BLOB	➡	BLOB a recibir la lista jerárquica
*	Operador	➡	* añadir el valor

Descripción

El comando **LIST TO BLOB** almacena la lista jerárquica *list* en el BLOB *blob*.

Si especifica el parámetro opcional *, la lista jerárquica se añade al final del BLOB y el tamaño del BLOB se extiende en consecuencia. Utilizando el parámetro opcional *, usted puede almacenar secuencialmente todo número de variables o listas (ver otros comandos BLOB) en un BLOB, siempre y cuando haya memoria suficiente.

Si no especifica el parámetro opcional *, la lista jerárquica se almacena al inicio del BLOB, reescribiendo su contenido anterior; el tamaño del BLOB se ajusta en consecuencia.

Cualquiera que sea el lugar donde la lista esté almacenada, el tamaño del BLOB aumentará si es necesario de acuerdo a la ubicación especificada (hasta el tamaño de la lista si es necesario). Los bytes modificados (diferentes de los que usted definió) se inicializan en 0 (cero).

Advertencia: si utiliza un BLOB para almacenar listas, luego debe llamar el comando **BLOB to list** para releer el contenido del BLOB, porque las listas son almacenadas en BLOBs utilizando un formato interno 4D.

Después de llamar el comando, si la lista se ha guardado correctamente, la variable OK toma el valor 1. Si la operación no pudo efectuarse, la variable OK toma el valor 0; por ejemplo, si no hubiera suficiente memoria disponible.

Nota sobre la independencia de plataforma: LIST TO BLOB y **BLOB to list** utilizan un formato interno 4D para administrar listas almacenadas en BLOBs. La ventaja es que usted no tiene que preocuparse por la conversión de bytes (byte swapping) entre plataformas cuando utiliza estos dos comandos. En otras palabras, un BLOB creado en Windows utilizando estos comandos puede ser reutilizado en Macintosh, y viceversa.

Ejemplo

Ver el ejemplo del comando **BLOB to list**.

LONGINT TO BLOB

LONGINT TO BLOB (enteroLargo ; BLOB ; byteOrden {; offset | *})

Parámetro	Tipo		Descripción
enteroLargo	Entero largo	➡	Valor de tipo Entero largo a escribir en el BLOB
BLOB	BLOB	➡	BLOB a recibir el valor Entero largo
byteOrden	Entero largo	➡	0 Orden de bytes nativo 1 Orden de bytes Macintosh 2 Orden de bytes PC
offset | *	Variable, Operador	➡	Offset en el BLOB (expresado en bytes) o * para añadir el valor
		⬅	Nuevo offset después de la escritura si se omite *

Descripción

El comando **LONGINT TO BLOB** escribe el valor de tipo Entero largo (4 bytes) *enteroLargo* en el BLOB *blob*.

El parámetro *byteOrder* fija el orden de los bytes ("byte ordering") del valor Entero largo (4 bytes) a escribir. Se pasa una de las siguientes constantes predefinidas de 4D:

Constante	Tipo	Valor
Native byte ordering	Entero largo	0
Macintosh byte ordering	Entero largo	1
PC byte ordering	Entero largo	2

Nota sobre la independencia de plataforma: si intercambia BLOBs entre las plataformas Macintosh y PC, es su decisión administrar los temas de byte swapping cuando utilice este comando.

Si especifica el parámetro opcional *, el valor entero largo (4 bytes) se añade al BLOB y el tamaño del BLOB se extiende en consecuencia. Utilizando el parámetro opcional *, puede almacenar secuencialmente cualquier número de valores de tipo *Entero*, *Entero largo*, *Real* o *Texto* (ver otros comandos de BLOB) en un BLOB, siempre y cuando haya memoria disponible.

Si no especifica el parámetro opcional * ni la variable del parámetro *offset*, el valor entero largo (4 bytes) se almacena al inicio del BLOB, reemplazando su contenido anterior; el tamaño del BLOB se ajusta en consecuencia.

Si pasa una variable en el parámetro *offset*, el valor entero largo (4 byte) se escribe a partir del offset (comenzando desde cero) en el BLOB. Sin importar donde escriba el valor entero largo (4 bytes), el tamaño del BLOB se incrementa de acuerdo a la ubicación que pase (más hasta 4 bytes, si es necesario). Los bytes redefinidos, diferentes de los que está escribiendo, se inicializan en cero.

Después de llamar el comando, se devuelve la variable del parámetro *offset*, incrementada en el número de bytes que hayan sido escritos. Por lo tanto, puede reutilizar esa misma variable con otro comando de escritura de BLOB para escribir otro valor.

Ejemplo 1

Después de la ejecución de este código:

```
LONGINT TO BLOB(0x01020304;vxBlob;Native byte ordering)
```

- El tamaño de *vxBlob* es 4 bytes
- En PowerPC *vxBLOB{0}=$01*, *vxBLOB{1}=$02*, *vxBLOB{2}=$03*, *vxBLOB{3}=$04*
- En Intel:PC *vxBLOB{0}=$04*, *vxBLOB{1}=$03*, *vxBLOB{2}=$02*, *vxBLOB{3}=$01*

Ejemplo 2

Después de la ejecución de este código:

```
LONGINT TO BLOB(0x01020304;vxBlob;Macintosh byte ordering)
```

- El tamaño de *vxBlob*es 4 bytes
- En todas las plataformas *vxBLOB{0}=$01*, *vxBLOB{1}=$02*, *vxBLOB{2}=$03*, *vxBLOB{3}=$04*

Ejemplo 3

Después de la ejecución de este código:

```
LONGINT TO BLOB(0x01020304;vxBlob;PC byte ordering)
```

- El tamaño de *vxBlob* es 4 bytes
- En todas las plataformas *vxBLOB{0}=$04*, *vxBLOB{1}=$03*, *vxBLOB{2}=$02*, *vxBLOB{3}=$01*

Ejemplo 4

Después de la ejecución de este código:

```
SET BLOB SIZE(vxBlob;100)
LONGINT TO BLOB(0x01020304;vxBlob;PC byte ordering;*)
```

- El tamaño de *vxBlob* es 104 bytes
- En todas las plataformas *vxBLOB{100}=$04*, *vxBLOB{101}=$03*, *vxBLOB{102}=$02*, *vxBLOB{103}=$01*
- Los otros bytes del BLOB no cambian

Ejemplo 5

Después de la ejecución de este código:

```
SET BLOB SIZE(vxBlob;100)
vlOffset:=50
LONGINT TO BLOB(0x01020304;vxBlob;Macintosh byte ordering;vlOffset)
```

- El tamaño de *vxBlob* es 100 bytes
- En todas las plataformas *vxBLOB{50}=$01*, *vxBLOB{51}=$02*, *vxBLOB{52}=$03*, *vxBLOB{53}=$04*
- Los otros bytes del BLOB no cambian
- La variable *vlOffset* se ha incrementado en 4 (y ahora es igual a 54)

REAL TO BLOB

REAL TO BLOB (real ; BLOB ; formatoReal {; offset | *})

Parámetro	Tipo		Descripción
real	Real	➡	Valor de tipo real a escribir en el BLOB
BLOB	BLOB	➡	BLOB a recibir el valor Real
formatoReal	Entero largo	➡	0 Formato real nativo 1 Formato real extendido 2 Formato real doble Macintosh 3 Formato real doble Windows
offset | *	Variable, Operador	➡	Offset en el BLOB (expresado en bytes) o * para añadir el valor
		⬅	Nuevo offset después de la escritura si se omite *

Descripción

El comando **REAL TO BLOB** escribe el valor de tipo Real (o numérico) *real* en el BLOB *blob*.

El parámetro *formatoReal* fija el formato interno y el orden de los bytes ("byte ordering") del valor de tipo Real a escribir. Usted pasa uno de las siguientes constantes predefinidas por 4D:

Constante	Tipo	Valor
Native real format	Entero largo	0
Extended real format	Entero largo	1
Macintosh double real format	Entero largo	2
PC double real format	Entero largo	3

Nota sobre la independencia de plataforma: si intercambia los BLOBs entre las plataformas Macintosh y PC, es su decisión administrar los temas de formatos reales y byte swapping cuando utilice este comando.

Si especifica el parámetro opcional *, el valor real se adjunta al BLOB; el tamaño del BLOB se extiende en consecuencia. Utilizando el parámetro * opcional, usted puede almacenar secuencialmente cualquier número de valores de tipo *Entero*, *Entero largo*, **Real** o *Texto* (ver otros comandos BLOB) en un BLOB, siempre y cuando tenga memoria disponible.

Si no especifica el parámetro opcional * ni la variable en el parámetro *offset*, el valor Real se almacena al comienzo del BLOB, reemplazando su contenido anterior; el tamaño del BLOB se ajusta en consecuencia.

Si pasa la variable en el parámetro *offset*, el valor Real se escribe en el offset (comenzando desde cero) en el BLOB. Sin importar donde escribe el valor Real, el tamaño del BLOB aumenta de acuerdo a la ubicación pasada (más hasta 8 ó 10 bytes, si es necesario). Los bytes redefinidos, diferentes a los que está escribiendo, se inicializan en cero.

Después de llamar el comando, se devuelve la variable del parámetro, incrementada en el número de bytes que hayan sido escritors. Por lo tanto, puede reutilizar la misma variable con otro comando de escritura de BLOB para escribir otro valor.

Ejemplo 1

Después de la ejecución de este código:

```
C_REAL(vrValor)
vrValor:=...
REAL TO BLOB(vrValor;vxBlob;Native real format)
```

- En todas las plataformas, el tamaño de *vxBlob* es 8 bytes

Ejemplo 2

Después de la ejecución de este código:

```
C_REAL(vrValor)
vrValor:=...
REAL TO BLOB(vrValor;vxBlob;Extended real format)
```

- En todas las plataformas, el tamaño de *vxBlob* is 10 bytes

Ejemplo 3

Después de la ejecución de este código:

```
C_REAL(vrValor)
vrValor:=...
REAL TO BLOB(vrValor;vxBlob;Macintosh double real format) ` o Formato real doble Windows
```

- En todas las plataformas, el tamaño de *vxBlob* es 8 bytes

Ejemplo 4

Después de la ejecución de este código:

```
SET BLOB SIZE(vxBlob;100)
C_REAL(vrValor)
vrValor:=...
INTEGER TO BLOB(vrValor;vxBlob;Windows Double real format)  ` o Formato real doble Macintosh
```

- En todas las plataformas, el tamaño de *vxBlob* es 8 bytes

Ejemplo 5

Después de la ejecución de este código:

```
SET BLOB SIZE(vxBlob;100)
REAL TO BLOB(vrValor;vxBlob;Extended real format;*)
```

- En todas las plataformas, el tamaño de *vxBlob* es 110 bytes
- En todas las plataformas, el valor real se almacena en los bytes #100 a #109
- Los otros bytes del BLOB no son modificados

Ejemplo 6

Después de la ejecución de este código:

```
SET BLOB SIZE(vxBlob;100)
C_REAL(vrValor)
vrValor:=...
vlOffset:=50
REAL TO BLOB(vrValor;vxBlob;Windows Double real format;vlOffset)  ` o Formato real doble Macintosh
```

- En todas las plataformas, el tamaño de *vxBlob* es 100 bytes
- En todas las plataformas, el valor real se almacena en los bytes #50 a #57
- Los otros bytes del BLOB se dejan sin cambiar
- La variable *vlOffset* se ha incrementado en 8 (y ahora es igual a 58)

SET BLOB SIZE

SET BLOB SIZE (BLOB ; tamaño {; relleno})

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	Campo o variable de tipo BLOB
tamaño	Entero largo	➡	Nuevo tamaño del BLOB
relleno	Entero largo	➡	Código ASCII del caracter de relleno

Descripción

SET BLOB SIZE redimensiona el BLOB *blob* de acuerdo al valor pasado en *tamaño*.
Si quiere asignar nuevos bytes a un BLOB y quiere inicializar estos bytes en un valor especifico, pase este valor (0..255) en el parámetro opcional *relleno*.

Ejemplo 1

Cuando usted ha terminado con un gran BLOB proceso o interproceso, es buena idea liberar la memoria que ocupa. Para hacer esto, escriba:

```
SET BLOB SIZE(aProcessBLOB;0)
SET BLOB SIZE(◇anInterprocessBLOB;0)
```

Ejemplo 2

El siguiente ejemplo crea un BLOB de 16K lleno de 0xFF:

```
C_BLOB(vxData)
SET BLOB SIZE(vxData;16*1024;0xFF)
```

Manejo de errores

Si no puede redimensionar un BLOB por memoria insuficiente, se genera el error -108. Puede interceptar este error utilizando el método de interrupción **ON ERR CALL**.

TEXT TO BLOB

TEXT TO BLOB (texto ; BLOB {; formatoTexto {; offset | *}})

Parámetro	Tipo		Descripción
texto	Cadena	➡	Texto a escribir en el BLOB
BLOB	BLOB	➡	BLOB a recibir el texto
formatoTexto	Entero largo	➡	Formato y conjunto de caracteres de texto
offset | *	Variable, Operador	➡	Offset en el BLOB (expresado en bytes) o * para añadir el valor
		⬅	Nuevo offset después de la escritura si se omite *

Descripción

El comando **TEXT TO BLOB** escribe el valor de tipo Texto *texto* en el BLOB *blob*.

El parámetro *formatoTexto* puede utilizarse para definir el formato interno y el conjunto de caracteres del valor de tipo Texto a escribir. Para hacer esto, pase una de las siguientes constantes (encontradas en el tema **BLOB**) en el parámetro *formatoTexto*:

Constante	Tipo	Valor
Mac C string	Entero largo	0
Mac Pascal string	Entero largo	1
Mac text with length	Entero largo	2
Mac text without length	Entero largo	3
UTF8 C string	Entero largo	4
UTF8 text with length	Entero largo	5
UTF8 text without length	Entero largo	6

Si omite el parámetro *formatoTexto*, 4D utiliza por defecto el formato **Mac C string**. En bases creadas a partir de la versión 11, 4D trabaja por defecto con el conjunto de caracteres Unicode (UTF8) para administrar texto, de manera que se recomienda utilizar este conjunto de caracteres. Por razones de compatibilidad, este comando puede utilizarse para "forzar" la conversion utilizando el conjunto de caracteres Mac Roman (utilizado en versiones anteriores de 4D). El conjunto de caracteres se elije vía el parámetro *formatoTexto.*

Notas:

- Las constantes "UTF8" sólo pueden utilizarse cuando la aplicación funciona en modo Unicode.
- La constante **Mac Text with length** no funciona con textos de más de 32 KB.
- Si quiere trabajar con conjuntos de caracteres diferentes de UTF8, utilice el comando CONVERT FROM TEXT.

La siguiente tabla describe cada uno de estos formatos:

Formato texto	Descripción y ejemplos
C string	El texto termina en un carácter NULL (código ASCII $00).
UTF8	"" $00
	"Café" $43 61 66 C3 A9 00
Mac	"" $00
	"Café" $43 61 66 8E 00
Pascal string	El texto está precedido de un byte de longitud.
UTF8	-
	-
Mac	"" $00
	"Café" $04 43 61 66 8E
Text with length	El texto está precedido por 4 bytes (UTF8) o 2 bytes (Mac) de longitud.
UTF8	"" $00 00 00 00
	"Café" $00 00 00 05 43 61 66 C3 A9
Mac	"" $00 00
	"Café" $00 04 43 61 66 8E
Text without length	El texto está compuesto únicamente por sus caracteres.
UTF8	"" Sin datos
	"Café" $43 61 66 C3 A9
Mac	"" Sin datos
	"Café" $43 61 66 8E

Si especifica el parámetro opcional *, el valor de tipo Texto se añade al BLOB; el tamaño del BLOB se extiende en consecuencia. Utilizando el parámetro opcional *, puede almacenar secuencialmente cualquier número de valores de tipo *Entero*, *Entero largo*, **Real** o *Texto* (ver otros comandos BLOB) en un BLOB, siempre y cuando haya memoria disponible.

Si no especifica el parámetro opcional * ni la variable del parámetro *offset*, el valor de tipo texto se almacena al comienzo del BLOB, reemplazando su contenido anterior; el tamaño del BLOB se ajusta en consecuencia.

Si pasa la variable en el parámetro *offset*, el valor de tipo Texto se escribe en el offset (a partir de cero) en el BLOB. Sin importar donde escriba el valor tipo Texto, el tamaño del BLOB aumenta de acuerdo a la ubicación pasada (más hasta el tamaño del texto, si es necesario). Los bytes redefinidos, diferentes de los que está escribiendo, se inicializan en cero.

Después de llamar el comando, la variable del parámetro *offset* se incrementa en el número de bytes que haya sido escrito. Por lo tanto, puede reutilizar la misma variable con otro comando de escritura de BLOB para escribir otro valor.

Ejemplo

Después de la ejecución este código:

```
SET BLOB SIZE(vxBlob;0)
C_TEXT(vtValor)
vtValor:="Café"  ` La longitud de vtValor es 4 bytes
TEXT TO BLOB(vtValor;vxBlob;Mac C string)  ` El tamaño del BLOB se vuelve 5 bytes
TEXT TO BLOB(vtValor;vxBlob;Mac Pascal string)  ` El tamaño del BLOB se vuelve 5 bytes
TEXT TO BLOB(vtValor;vxBlob;Mac text with length)  ` El tamaño del BLOB se vuelve 6 bytes
TEXT TO BLOB(vtValor;vxBlob;Mac text without length)  ` El tamaño del BLOB se vuelve 4 bytes
TEXT TO BLOB(vtValor;vxBlob;UTF8 C string)  ` El tamaño del BLOB se vuelve 6 bytes
TEXT TO BLOB(vtValor;vxBlob;UTF8 text with length)  ` El tamaño del BLOB se vuelve 9 bytes
TEXT TO BLOB(vtValor;vxBlob;UTF8 text without length)  ` El tamaño del BLOB se vuelve 5 bytes
```

VARIABLE TO BLOB

VARIABLE TO BLOB (variable ; BLOB {; offset | *})

Parámetro	Tipo		Descripción
variable	Variable	➡	Variable a guardar en el BLOB
BLOB	BLOB	➡	BLOB a recibir la variable
offset | *	Variable, Operador	➡	Offset de la variable (expresado en bytes) en el BLOB o * para añadir el valor
		⬅	Nuevo offset después de escritura si se omite *

Descripción

El comando **VARIABLE TO BLOB** almacena la variable *variable* en el BLOB *blob*.

Si especifica el parámetro opcional *, la variable se añade al BLOB y el tamaño del BLOB se extiende en consecuencia. Utilizando el parámetro opcional *, usted puede almacenar secuencialmente todo número de variables o listas (ver otros comandos BLOB) en un BLOB, el único limite es la memoria disponible.

Si no especifica el parámetro opcional * ni la variable en el parámetro *offset*, la variable se almacena al comienzo del BLOB, sobre su contenido anterior; el tamaño del BLOB se redimensiona en consecuencia.

Si pasa la variable *offset* en parámetro, la variable se escribe en BLOB al offset (a partir de cero). Sin importar donde escribe la variable, el tamaño del BLOB aumenta de acuerdo a la ubicación pasada (más el tamaño de la variable, si es necesario). Los nuevos bytes redefinidos, diferentes de los bytes en los que está escribiendo, se inicializan en cero.

Después de llamar el comando, la variable del parámetro *offset* se incrementa en el número de bytes escritos. Por lo tanto, usted puede reutilizar la misma variable con otro comando de escritura de BLOB para poner otra variable o lista.

VARIABLE TO BLOB acepta todo tipo de variables (incluyendo otros BLOBs), excepto los siguientes:

- Puntero
- Array de punteros

Note que:

- si guarda una variable de tipo Entero largo que es una referencia a una lista jerárquica (ListRef), **VARIABLE TO BLOB** guardará la variable Entero largo, no la lista. Para guardar y recuperar las listas jerárquicas en un BLOB, utilice los comandos **LIST TO BLOB** y **BLOB to list**.
- si pasa un objeto **C_OBJECT** o una colección **C_COLLECTION**) en el parámetro *variable*, el comando ubica una copia (y no una referencia) en el BLOB como JSON en UTF-8. Si el objeto o colección contiene punteros, los valores no referenciados se guardan en el BLOB, no los punteros mismos.

Sin embargo, si almacena una variable de tipo *Entero largo* que es una referencia a una lista jerárquica (*ListRef*), **VARIABLE TO BLOB** almacenará la variable *Entero largo*, no la lista. Para guardar y recuperar las listas jerárquicas en un BLOB, utilice los comandos **LIST TO BLOB** y **BLOB to list**.

Advertencia: Si utiliza un BLOB para almacenar las variables, debe utilizar posteriormente el comando **BLOB TO VARIABLE** para recuperar el contenido del BLOB, porque las variables son almacenadas en BLOBs utilizando un formato interno de 4D.

Después del llamar al comando, si la variable ha sido almacenada con éxito, la variable OK toma el valor 1. Si la operación no se pudo realizar, la variable OK toma el valor 0; por ejemplo, por falta de memoria.

Nota sobre la independencia de la plataforma: VARIABLE TO BLOB y **BLOB TO VARIABLE** utilizan un formato interno de 4D para administrar las variables almacenadas en los BLOBs. La ventaja es que usted no tiene que preocuparse por la conversión de bytes (byte swapping) entre plataformas mientras utiliza estos dos comandos. En otras palabras, un BLOB creado en Windows utilizando cualquiera de estos dos comandos puede ser reutilizado en Macintosh y viceversa.

Ejemplo 1

Los siguientes métodos de proyecto le permiten almacenar y recuperar rápidamente las arrays en los documentos en disco:

```
  ` Método de proyecto GUARDAR ARRAY
  ` GUARDAR ARRAY ( Alfa; Puntero )
  ` GUARDAR ARRAY ( Documento ; -> Array )
C_STRING(255;$1)
C_POINTER($2)
C_BLOB($vxArrayDatos)
VARIABLE TO BLOB($2->;$vxArrayDatos)  ` Almacenar el array en el BLOB
COMPRESS BLOB($vxArrayDatos)  ` Comprimir el BLOB
BLOB TO DOCUMENT($1;$vxArrayDatos)  ` Guardar el BLOB en disco

  ` Método de proyecto CARGAR ARRAY
  ` CARGAR ARRAY ( Alfa ; Puntero )
  ` CARGAR ARRAY ( Documento ; -> Array )
C_STRING(255;$1)
C_POINTER($2)
C_BLOB($vxArrayDatos)
DOCUMENT TO BLOB($1;$vxArrayDatos)  ` Cargar el BLOB del disco
EXPAND BLOB($vxArrayDatos)  ` Descomprimir el BLOB
BLOB TO VARIABLE($vxArrayDatos;$2->)  ` Recuperar el array del BLOB
```

Después de añadir estos métodos a su aplicación, puede escribir:

```
ARRAY STRING(...;comoTodoArray;...)
  ` ...
GUARDAR ARRAY($vsDocNombre;->comoTodoArray)
  ` ...
CARGAR ARRAY($vsDocNombre;->comoTodoArray)
```

Ejemplo 2

Los dos métodos de proyecto siguientes permiten almacenar y recuperar rápidamente las variables en un BLOB:

```
  ` Método de proyecto GUARDAR VARIABLES EN BLOB
  ` GUARDAR VARIABLES EN BLOB ( Puntero{ ; Puntero... { ; Puntero} } )
  ` GUARDAR VARIABLES EN BLOB ( BLOB { ; Var1 ... { ; Var2 } } )
C_POINTER($1)
C_LONGINT($vlParam)

SET BLOB SIZE($1->;0)
For($vlParam;2;Count parameters)
   VARIABLE TO BLOB($[$vlParam]->;$1->;*)
End for


  ` Método de proyecto RECUPERAR VARIABLES DEL BLOB
  ` RECUPERAR VARIABLES DEL BLOB ( Puntero{ ; Puntero... { ; Puntero} } )
  ` RECUPERAR VARIABLES DEL BLOB ( BLOB { ; Var1 ... { ; Var2 } } )
C_POINTER($1)
C_LONGINT($vlParam;$vlOffset)

$vlOffset:=0
For($vlParam;2;Count parameters)
   BLOB TO VARIABLE($1->;$[$vlParam]->;$vlOffset)
End for
```

Después de escribir estos métodos en su aplicación, puede escribir:

```
GUARDAR VARIABLES EN BLOB(->vxBLOB;->vgImagen;->comoUnArray;->alOtroArray)
  ` ...
RECUPERAR VARIABLES DEL BLOB(->vxBLOB;->vgImagen;->comoUnArray;->alOtroArray)
```

Variables y conjuntos del sistema

La variable OK toma el valor 1 si la variable ha sido almacenada correctamente, de lo contrario toma el valor 0.

Booleanos

- Comandos Booleanos
- Bool
- False
- Not
- True

Comandos Booleanos

4D incluye funciones Booleanas, utilizadas para cálculos Booleanos:

```
True
False
Not
```

Ejemplo

Este ejemplo define una variable **Booleana** basada en el valor de un botón. La variable devuelve True en *miBooleano* si se hace clic en el botón *miBoton y* False si no se hace clic en el botón. Cuando un botón recibe un clic, la variable del botón toma el valor 1.

```
If(miBoton=1)  ` Si el botón recibió un clic
   miBooleano:=True  ` miBooleano toma el valor True
Else  ` Si el botón no recibió un clic,
   miBooleano:=False  ` miBooleano toma el valor False
End if
```

El ejemplo anterior se puede simplificar en una línea.

```
miBooleano:=(miBoton=1)
```

Bool

Bool (expresion) -> Resultado

Parámetro	Tipo		Descripción
expresion	Expresión	➡	Expresión para la cual devolver la forma booleana
Resultado	Booleano	⮌	Forma booleana de la expresión

Descripción

El comando **Bool** devuelve la forma booleana de la expresión que se pasó en *expresion*.

El comando puede devolver los siguientes valores, dependiendo del tipo de resultado de la *expresion*:

Tipo de la expresión	Retorno del comando Bool
Indefinido	False
Nulo	False
Booleano	False si falso, de lo contrario True
Número	False si 0, otro True
Otros tipos	False

Este comando es útil cuando se espera que el resultado de una expresión sea un booleano, cualquiera que sea el resultado real de su evaluación (por ejemplo, si se evalúa como **nulo** o **indefinido**).

Ejemplo

Selecciona un valor dependiendo del contenido de un atributo de campo de objeto, anticipando el caso en el que falta el atributo:

```
C_TEXT($married)
$married:=Choose(Bool([Person]data.married);"Married";"Single")
 //"Single" si el atributo "married" no se encuentra en el campo
ALERT("This person is "+$married)
```

False

False -> Resultado

Parámetro	Tipo	Descripción
Resultado	Booleano	False

Descripción

False devuelve el valor booleano Falso (False).

Ejemplo

El siguiente ejemplo asigna la variable *vbOpciones* a Falso:

```
vbOpciones:=False
```

Not

Not (booleano) -> Resultado

Parámetro	Tipo		Descripción
booleano	Booleano	⇒	Valor booleano a negar
Resultado	Booleano	⇐	Opuesto del booleano

Descripción

La función **Not** devuelve la negación del *booleano*, cambiando Verdadero por Falso o Falso por Verdadero.

Ejemplo

Este ejemplo asigna primero Verdadero a una variable, luego cambia el valor de la variable a Falso, y luego nuevamente a Verdadero.

```
vResultado:=True  ` vResultado toma el valor Verdadero
vResultado:=Not(vResultado)  ` vResultado toma el valor Falso
vResultado:=Not(vResultado)  ` vResultado toma el valor Verdadero
```

True

True -> Resultado

Parámetro	Tipo	Descripción
Resultado	Booleano	True

Descripción

True devuelve el valor booleano Verdadero (True).

Ejemplo

El siguiente ejemplo asigna la variable *vbOpciones* a Verdadero :

```
vbOpciones:=True
```

Búsquedas

- DESCRIBE QUERY EXECUTION
- Find in field
- Get last query path
- Get last query plan
- GET QUERY DESTINATION
- Get query limit
- ORDER BY
- ORDER BY ATTRIBUTE
- ORDER BY FORMULA
- QUERY
- QUERY BY ATTRIBUTE
- QUERY BY EXAMPLE
- QUERY BY FORMULA
- QUERY SELECTION
- QUERY SELECTION BY ATTRIBUTE
- QUERY SELECTION BY FORMULA
- QUERY SELECTION WITH ARRAY
- QUERY WITH ARRAY
- SET QUERY AND LOCK
- SET QUERY DESTINATION
- SET QUERY LIMIT

DESCRIBE QUERY EXECUTION

DESCRIBE QUERY EXECUTION (estado)

Parámetro	Tipo		Descripción
estado	Booleano	➡	True=Activar análisis de búsquedas internas, False=Desactivar el análisis de búsquedas internas

Descripción

El comando **DESCRIBE QUERY EXECUTION** permite activar o desactivar el modo de análisis de la ejecución de búsquedas para el proceso actual. El comando funciona únicamente en el contexto de los comandos de búsqueda del lenguaje 4D tal como **QUERY**.

La llamada del comando con el parámetro *estado* en **True** activa el modo del análisis de búsquedas. En este modo, el motor de 4D registra internamente dos series de informaciones específicas para cada búsqueda posterior efectuada sobre los datos:

- Una descripción interna detallada de la búsqueda justo antes de su ejecución, en otras palabras, la búsqueda previa (el plan de búsqueda),
- Una descripción interna detallada de la búsqueda que se ejecutó realmente (la ruta de búsqueda).

La información registrada incluye el tipo de búsqueda (indexada, secuencial), el número de registro encontrados y el tiempo necesario para cada criterio de búsqueda a ejecutar. Puede leer esta información utilizando los comandos **Get last query plan** y **Get last query path**.

Por lo general, la descripción del plan de una búsqueda y su ruta son idénticos, pero podrían eventualmente ser diferentes porque 4D podría implementar optimizaciones dinámicas durante la ejecución de la búsqueda para mejorar el rendimiento. Por ejemplo, una búsqueda índexada puede convertirse dinámicamente en una búsqueda secuencial si el motor 4D estima que sería más rápida, este es el caso, cuando el número de registros en los cuales se efectúa la búsqueda es bajo.

Pase **False** en el parámetro *estado* cuando no necesite analizar las búsquedas. El modo de análisis de la ejecución de las búsquedas puede volver lenta la aplicación.

Ejemplo

El siguiente ejemplo ilustra el tipo de información obtenida utilizando estos comandos:

```
C_TEXT($vResultPlan;$vResultPath)
DESCRIBE QUERY EXECUTION(True) //modo análisis
QUERY([Employees];[Employees]LastName="T@";*) // Búsqueda de los empleados cuyo apellido comienza por T...
QUERY([Employees]; & ;[Companies]Name="H@";*) // que trabajan para una empresa cuyo nombre comienza por H
QUERY([Employees]; & ;[Employees]Salary>2500;*) // cuyo salario es > 2500
QUERY([Employees]; & ;[Cities]Pop<50000) // que viven en una ciudad con menos de 50 000 habitantes
$vResultPlan:=Get last query plan(Description in text format)
$vResultPath:=Get last query path(Description in text format)
DESCRIBE QUERY EXECUTION(False) //Fin del modelo de análisis
```

Después de ejecutar este código, *$vResultPlan* y *$vResultPath* contienen descripciones de las búsquedas realizadas, por ejemplo:

```
$vResultPlan :
   Employees.LastName == T@ And Employees.Salary > 2500 And Join on Table : Companies : Employees.Company = Companies.Name
[index : Companies.Name ] LIKE H@ And Join on Table : Cities : Employees.City = Cities.Name [index : Cities.Pop ] < 50000
$vResultPath :
(Employees.LastName == T@ And Employees.Salary > 2500) And (Join on Table : Companies : Employees.Company =
Companies.Name with filter {[index : Companies.Name ] LIKE H@}) And (Join on Table : Cities : Employees.City = Cities.Name with filter
{[index : Cities.Pop ] < 50000}) (3 registros encontrados en 1 minuto)
```

Si la constante Description in XML Format se pasa al comando **Get last query path**, *$vResultPath* contiene la descripción de la búsqueda expresada en XML:

```
$vResultPath : <QueryExecution> <steps description="And" time="0" recordsfounds="1227"> <steps description="
[Merge] : ACTORS with CITIES" time="13" recordsfounds="1227"> <steps description="[Join] : ACTORS.Birth_City_ID
=CITIES.City_ID" time="13" recordsfounds="1227"/> </steps> </steps> </QueryExecution>
```

Find in field

Find in field (campoObjetivo ; valor) -> Resultado

Parámetro	Tipo		Descripción
campoObjetivo	Campo	➡	Campo objetivo en el cual ejecutar la búsqueda.
valor	Campo, Variable	➡	Valor a buscar
		⬅	Valor encontrado
Resultado	Entero largo	➲	Número del registro encontrado o -1 si no se encontró ningún registro

Descripción

El comando **Find in field** devuelve el número del primer registro cuyo *campoObjetivo* es igual a *valor*.
Si no se encuentran registros, **Find in field** devuelve -1.

Después de llamar este comando, *valor* contiene el valor encontrado. Esta funcionalidad le permite efectuar búsquedas utilizando el carácter ("@") en campos tipo Alfa y luego recuperar el valor encontrado.

Nota: por este principio, no se puede utilizar un parámetro ($1, $2, etc.) en *valor* porque esto causaría un mal funcionamiento en modo compilado. Del mismo modo, si pasa un campo en el parámetro *valor*, tenga en cuenta que su valor será reasignado si la consulta tiene éxito (el comando **Modified record**, en particular, devolverá True para el registro actual de la tabla).
Este comando no modifica la selección actual ni el registro actual.

Este comando es rápido y muy útil para evitar la creación de entradas dobles durante la entrada de datos.
Nota histórica: en versiones anteriores de 4D, el comando **Find in field** era llamado **Find index key** y sólo funcionaba con campos indexados. A partir de 4D v11 SQL, esta limitación se eliminó y el comando se renombró.

Ejemplo 1

En una base de datos de CDs, durante la entrada de datos asumamos que usted quiere verificar el nombre del cantante para ver si ya existe en la base. Como pueden existir homónimos, usted quiere que el campo [Cantante]Nombre sea único. Por lo tanto, en el formulario de entrada, puede escribir el siguiente código en el método de objeto del campo [Cantante]Nombre:

```
If(Form event=On Data Change)
   $RecNum:=Find in field([Cantante]Nombre;[Cantante]Nombre)
   If($RecNum #-1)  ` si este nombre ya ha sido introducido
      CONFIRM("Ya existe un cantante con el mismo nombre. ¿Quiere ver el registro?";"Sí";"No")
      If(OK=1)
         GOTO RECORD([Cantante];$RecNum)
      End if
   End if
End if
```

Ejemplo 2

Este es un ejemplo que permite verificar la existencia de un valor:

```
C_LONGINT($id;$1)
$id:=$1
If(Find in field([MyTable]MyID;$id)>=0)
   $0:=True
Else
   $0:=False
End if
```

Tenga en cuenta >= permite cubrir todos los casos. De hecho, la función devuelve un número de registro y el primer registro tiene el número 0.

Get last query path

Get last query path (formatDesc) -> Resultado

Parámetro	Tipo		Descripción
formatDesc	Entero largo	⇒	Formato de descripción (Texto o XML)
Resultado	Cadena	⇐	Descripción de la ruta de la última búsqueda ejecutada

Descripción

El comando **Get Last Query Path** devuelve la descripción interna detallada de la ruta real de la última búsqueda efectuada en los datos. Para mayor información sobre las descripciones de búsquedas, consulte la documentación del comando **DESCRIBE QUERY EXECUTION**.

Esta descripción se devuelve en formato Texto o XML dependiendo del valor pasado en el parámetro *formatDesc*. Puede pasar una de las siguientes constantes, ubicadas en el tema "**Queries**":

Constante	Tipo	Valor
Description in text format	Entero largo	0
Description in XML format	Entero largo	1

Este comando devuelve un valor significativo si el comando **DESCRIBE QUERY EXECUTION** ha sido ejecutado durante la sesión.

La descripción de la ruta de la última búsqueda puede compararse con la descripción del plan de búsqueda de la última búsqueda (obtenido utilizando el comando **Get Last Query Plan**) con propósitos de optimización.

Get last query plan

Get last query plan (formatDesc) -> Resultado

Parámetro	Tipo		Descripción
formatDesc	Entero largo	→	Formato de descripción (Texto o XML)
Resultado	Cadena	←	Descripción del plan de la última búsqueda ejecutada

Descripción

El comando **Get Last Query Plan** devuelve la descripción interna del plan de ejecución de la última búsqueda llevada a cabo en los datos. Para mayor información sobre las descripciones de búsquedas, por favor consulte la documentación del comando **DESCRIBE QUERY EXECUTION**.

Esta descripción se devuelve en formato Texto o XML dependiendo del valor pasado en el parámetro *formatDesc*. Puede pasar una de las siguientes constantes, ubicadas en el tema "**Queries**":

Constante	Tipo	Valor
Description in text format	Entero largo	0
Description in XML format	Entero largo	1

Este comando devuelve un valor significativo si el comando **DESCRIBE QUERY EXECUTION** ha sido ejecutado durante la sesión.

La descripción del plan de la última búsqueda puede compararse con la descripción de la ruta real de la última búsqueda (obtenido con la ayuda del comando **Get Last Query Path**) con propósitos de optimización.

GET QUERY DESTINATION

GET QUERY DESTINATION (destinoTipo ; destinoObjeto ; destinoPunt)

Parámetro	Tipo		Descripción
destinoTipo	Entero largo	⬅	0=selección actual, 1=conjunto, 2=selección temporal, 3=variable
destinoObjeto	Cadena	⬅	Nombre del conjunto o Nombre de la selección temporal o Cadena vacía
destinoPunt	Puntero	⬅	Puntero a la variable local si destinoTipo=3

Descripción

El comando **GET QUERY DESTINATION** devuelve el destino actual de los resultados de las búsquedas para el proceso en curso. Por defecto, los resultados de las búsquedas modifican la selección actual, pero puede modificar este funcionamiento con la ayuda del comando **SET QUERY DESTINATION**.

En el parámetro *destinoTipo*, 4D devuelve un valor indicando el destino actual de las búsquedas y en el parámetro *destinoObjeto* devuelve el nombre del destino (si aplica). Puede comparar el valor del parámetro *destinoTipo* con las constantes del tema **Destinos de búsqueda**:

Constante	Tipo	Valor
Into current selection	Entero largo	0
Into named selection	Entero largo	2
Into set	Entero largo	1
Into variable	Entero largo	3

El valor devuelto en el parámetro *destinoObjeto* depende del valor del parámetro *destinoTipo*:

Parámetro destinoTipo	Parámetro destinoObjeto
0 (selección actual)	*destinoObjeto* es una cadena vacía
1 (conjunto)	*destinoObjeto* contiene el nombre del conjunto
2 (selección temporal)	*destinoObjeto* contiene el nombre de la selección
3 (variable)	*destinoObjeto* es una cadena vacía (utilizar el parámetro *destinoPunt*)

Cuando el destino de las búsquedas es una variable local (*destinoTipo* devuelve 3), 4D devuelve en el parámetro *destinoPunt* un puntero a esta variable.

Ejemplo

Queremos modificar temporalmente el destino de búsqueda y restablecer los parámetros previos:

```
GET QUERY DESTINATION($vType;$vName;$ptr)
 //recuperación de los parámetros actuales
SET QUERY DESTINATION(Into set;"$temp")
 //modificación temporal del destino
QUERY(...) //búsqueda
SET QUERY DESTINATION($vType;$vName;$ptr)
 //restablecimiento de los parámetros
```

Get query limit

Get query limit -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Número límite de registros, 0 = número ilimitado

Descripción

El comando **Get query limit** devuelve el límite del número de registros que una búsqueda puede encontrar en el proceso actual.
Se define este límite utilizando el comando **SET QUERY LIMIT**.
Por defecto, si ningún límite se ha definido, el comando devuelve 0.

ORDER BY

ORDER BY ({tabla ;}{ unCampo }{; > o < }{; unCampo2 ; > o <2 ; ... ; unCampoN ; > o <N}{; *})

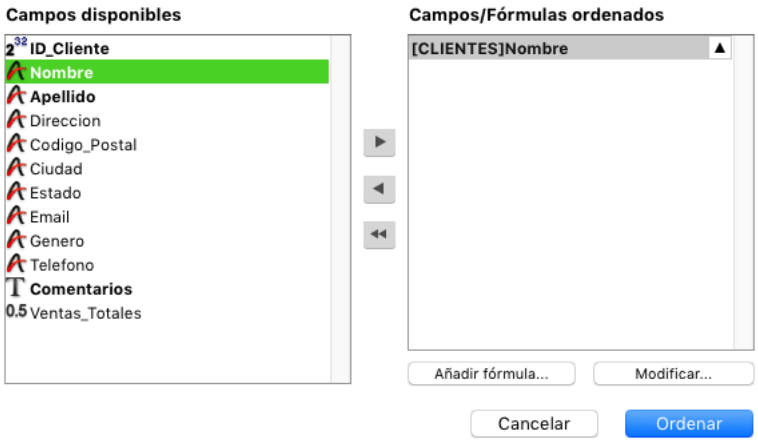
Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual ordenar los registros seleccionados o Tabla por defecto si se omite
unCampo	Campo	➡	Campo en el cual efectuar la ordenación para cada nivel
> o <	Operador	➡	Sentido de la ordenación para cada nivel: > para orden ascendente o < para orden descendente
*	Operador	➡	Continua la bandera de ordenación

Descripción

ORDER BY ordena (reordena) los registros de la selección actual de *tabla* para el proceso actual. Una vez efectuada la ordenación, el primer registro de la nueva selección actual se convierte en el registro actual.

Si omite el parámetro *tabla*, el comando se aplica a la tabla por defecto, si se ha definido una tabla por defecto. Si no 4D utiliza la tabla del primer campo pasado como parámetro. Si no pasa un parámetro y si no se ha definido una tabla por defecto, se devuelve un error.

Si no especifica el parámetro *campo*, ni los parámetros *> o <* o, ORDER BY muestra la caja de diálogo Ordenar de 4D para *tabla*. Esta es la caja de diálogo del editor:



Para mayor información sobre la utilización de este editor, consulte el Manual de diseño de 4D.

Si especifica los parámetros *unCampo* y *> o <*, la caja de diálogo estándar de Ordenar no se muestra y la ordenación se define por programación. Puede ordenar la selección en un nivel o en varios niveles. Para cada nivel de clasificación, se especifica un campo en *unCampo* y el orden de clasificación en *> o <*. Si pasa el símbolo "mayor que" (>), el orden es ascendente. Si pasa el símbolo "menor que" (<), el orden es descendente.

Si omite el parámetro de ordenación *> o <*, el orden es ascendente por defecto.

Si sólo se especifica un campo (ordenación de un nivel) y se indexa, se utiliza el índice para el orden. Si el campo no está indexado o si hay más de un campo, el orden se ejecuta secuencialmente (excepto en el caso de índices compuestos). El campo puede pertenecer a la tabla de la selección que está siendo reordenada o a una tabla 1 relacionada con *tabla* por una relación automática. En este caso, el orden es siempre secuencial.
Si los campos ordenados están incluidos en un índice compuesto, **ORDER BY** utiliza el índice para el orden.

Para ordenaciones múltiples (ordenar en varios campos), puede llamar**ORDER BY** tantas veces como sea necesario y especificar el parámetro opcional *, excepto para la última llamada **ORDER BY**, que inicia la operación de ordenación real. Esta funcionalidad es útil para la gestión de ordenaciones multicriterios en interfaces de usuario personalizadas.
Atención: con esta sintaxis, sólo puede pasar un nivel de ordenación (campo) por línea de instrucción.

No importa cómo se haya definido una ordenación, si la operación de ordenación real va a tomar algún tiempo para realizarse, 4D muestra automáticamente un mensaje que contiene un termómetro de progreso. Estos mensajes se pueden activar y desactivar utilizando los comandos **MESSAGES ON** y **MESSAGES OFF**. Si se muestra el termómetro de progreso, el usuario puede hacer clic en el botón Detener para interrumpir la ordenación.

Si la ordenación se realiza correctamente, la variable OK toma el valor *1*. Si el usuario hace clic en Cancelar, **ORDER BY** termina sin efectuar la ordenación, y la variable OK toma el valor *0* (cero).

Nota: este comando no soporta campos de tipo Objeto.

Ejemplo 1

El siguiente ejemplo muestra la caja de diálogo Ordenar para la tabla [Productos]:

```
ORDER BY([Productos])
```

Ejemplo 2

El siguiente ejemplo muestra la caja de diálogo Ordenar para la tabla por defecto (si ha sido definida):

```
ORDER BY
```

Ejemplo 3

El ejemplo siguiente ordena la selección actual de [Productos] por nombre en orden ascendente:

```
ORDER BY([Productos];[Productos]Nombre;>)
```

Ejemplo 4

El siguiente ejemplo ordena la selección actual de [Productos] por nombre en orden descendente:

```
ORDER BY([Productos];[Productos]Nombre;<)
```

Ejemplo 5

La línea siguiente ordena la selección de [Productos] por tipo y precio en orden ascendente para ambos niveles:

```
ORDER BY([Productos];[Productos]Tipo;>;[Productos]Precio;>)
```

Ejemplo 6

El siguiente ejemplo ordena la selección actual de [Productos] por tipo y precio en orden descendente para ambos niveles:

```
ORDER BY([Productos];[Productos]Tipo;<;[Productos]Precio;<)
```

Ejemplo 7

El siguiente ejemplo ordena la selección actual de [Productos] por tipo en orden ascendente y por precio en orden descendente:

```
ORDER BY([Productos];[Productos]Tipo;>;[Productos]Precio;<)
```

Ejemplo 8

El siguiente ejemplo ordena la selección actual de [Products] por tipo en orden descendente y por precio en orden ascendente:

```
ORDER BY([Products];[Products]Type;<;[Products]Price;>)
```

Ejemplo 9

El siguiente ejemplo efectúa una ordenación indexada si el campo [Productos]Nombre está indexado:

```
ORDER BY([Productos];[Productos]Nombre;>)
```

Ejemplo 10

El siguiente ejemplo ordena la selección actual de [Products] por nombre en orden ascendente:

```
ORDER BY([Products];[Products]Name
```

Ejemplo 11

El siguiente ejemplo efectúa una ordenación secuencial, sin importar si los campos están indexados:

```
ORDER BY([Productos];[Productos]Tipo;>;[Productos]Precio;>)
```

Ejemplo 12

La siguiente línea realiza una ordenación secuencial utilizando un campo relacionado:

```
SET FIELD RELATION([Employee]Company_ID;Automatic;Do not modify)
ORDER BY([Employee];[Company]LastName)
SET FIELD RELATION([Employee]Company_ID;Structure configuration;Do not modify)
```

Ejemplo 13

El siguiente ejemplo realiza una ordenación indexada en dos niveles si un índice compuesto [Contacts]LastName + [Contacts]FirstName cse ha especificado en la base:

```
ORDER BY([Contacts];[Contacts]LastName;>;[Contacts]FirstName;>)
```

Ejemplo 14

En un formulario de salida mostrado en modo Aplicación, usted le permite a los usuarios ordenar una columna en orden creciente simplemente haciendo clic en el encabezado de la columna. Si el usuario mantiene presionada la tecla **Mayús** mientras hace clic en otros encabezados de columnas, la ordenación se lleva a cabo en varios niveles:

Título Álbum :	Intérprete :	Formato :	Categoría :
Nat King Cole's Greatest Love Songs	Nat King Cole	CD	Ambiente
Johnny Mathis, 16 Most Requested Songs	Johnny Mathis	CD	Ambiente
Carpenters - Their Greatest Hits	Carpenters, The	CD	Ambiente
Whitney Houston	Whitney Houston	CD	Ambiente
Johnny Mathis: In The Still Of The Night	Johnny Mathis	CD	Ambiente
Best of B. B. King	B. B. King	DVD	Blues
Virtuoso - Ludwig Van Beethoven	Berliner Philharmoniker	CD	Clásica
Brahms Piano Quintet - Clarinet Quintet	Benda Musicians, The	CD	Clásica
Season for Love	London Symphony Orchestra	CD	Clásica
Lucille and Other Classics by Kenny Rogers	Kenny Rogers	CD	Country
Jazzis Magazine April 1995 Collection	Various	CD	Jazz
The Long Run	Eagles	CD	Rock
The Best of the Sylistics	Stylistics, The	Casete	Soul
Temptations 25th Anniversary Volume II	Temptations, The	CD	Soul
Best of Gladys Knight & the Pips, 1973-1988	Gladys Knight & the Pips	Casete	Soul
Bad	Michael Jackson	Video	Soul

Cada encabezado de columna contiene un botón resaltado asociado con el siguiente método de objeto:

```
MULTILEVEL(->[CDs]Title)  `Botón del encabezado de la columna título
```

Cada botón llama al método de proyecto MULTINIVEL con un puntero al campo de la columna correspondiente. El método proyecto MULTINIVEL es el siguiente:

```
  ` Método proyecto MULTINIVEL
  ` MULTILEVEL (Pointer)
  ` MULTILEVEL (->[Table]Field)

C_POINTER($1)  `Nivel de ordenación (campo)
C_LONGINT($lLevelNb)

  `Construcción de criterios
If(Not(Shift down))  `Ordenación simple (un nivel)
   ARRAY POINTER(aPtrSortField;1)
   aPtrSortField{1}:=$1
Else
   $lLevelNb:=Find in array(aPtrSortField;$1)  `¿Ya está ordenado este campo?
   If($lLevelNb<0)  `Si no
      INSERT IN ARRAY(aPtrSortField;Size of array(aPtrSortField)+1;1)
      aPtrSortField{Size of array(aPtrSortField)}:=$1
   End if
End if
  `Ejecución de la ordenación
$lLevelNb:=Size of array(aPtrSortField)
If($lLevelNb>0)  `Hay por lo menos un nivel de ordenación
   For($i;1;$lLevelNb)
      ORDER BY([CDs];(aPtrSortField{$i})->;>;*)  `Construir la ordenación
   End for
```

```
    ORDER BY([CDs]) `No * termina la definición de la ordenación y comienza la operación de ordenación actual.
End if
```

ORDER BY ATTRIBUTE

ORDER BY ATTRIBUTE ({tabla ;} campoObjeto ; rutaAtrib ; > o < {; campoObjeto2 ; rutaAtrib2 ; > o <2 ; ... ; campoObjetoN ; rutaAtribN ; > o <N} {; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual ordenar los registros seleccionados, o tabla por defecto, si se omite
campoObjeto	Campo Objeto	➡	Campo objeto que contiene el atributo de ordenación
rutaAtrib	Cadena	➡	Nombre o ruta del atributo en el que se define el orden para cada nivel
> o <	Operador	➡	Dirección de ordenación para cada nivel: > para ordenar en orden ascendente o < para ordenar en orden descendente
*	Operador	➡	Continuar bandera de orden

Descripción

El comando **ORDER BY ATTRIBUTE** ordena (reordena) los registros de la selección actual de *tabla* para el proceso actual basado en el contenido de *rutaAtrib* de *campoObjeto*. Después de terminada la ordenación, el nuevo primer registro de la selección se convierte en el registro actual.

Si omite el parámetro *tabla*, el comando se aplica a la tabla por defecto, si se ha especificado. De lo contrario, 4D utiliza la tabla del primer campo pasado como un parámetro.

En *campoObjeto*, pase el campo Objeto cuyo atributo desea utilizar para la ordenación. Este campo puede pertenecer a *tabla* o a una tabla relacionada con *tabla* con una relación automática o manual. En este caso, la ordenación es siempre secuencial.

En *rutaAtrib*, pase la ruta del atributo cuyos valores desea utilizar para ordenar los registros, por ejemplo "niños.genero.edad".

Notas:

- Sólo los atributos que contienen valores escalares (números, textos, booleanos, fecha) pueden ser ordenados. Otros tipos de valores (objeto, imagen...) son considerados como valores nulos.
- No se puede pasar un elemento de un array en rutaAtrib (en este caso, se devuelve un error).
- Recuerde que los nombres de atributos son sensibles a las mayúsculas y minúsculas: "MiAtt" y "miAtt" son nombres de atributos diferentes en el mismo registro.

Si el atributo del campo contiene valores de tipos diferentes (es decir, números, cadenas, booleanos), primero se agrupan por tipo, luego por valor.

Si el valor del atributo de campo es **nulo** para algunos registros (es decir, el valor del atributo es nulo o *rutaAtrib* no existe en el campo):

- Si la orden es **ascendente** (>), los registros con valor **nulo** se colocarán al principio de la selección.
- Si el orden es **descendente** (<), los registros con valor **nulo** se colocarán al final de la selección.

Puede ordenar la selección en uno o en varios niveles. Para cada nivel de ordenación, se especifica un *campo*, un *rutaAtrib* y el sentido de ordenación en *> o <*. Si pasa el símbolo (>) "mayor que", el orden es ascendente. Si pasa el símbolo (<) "menor que", el orden es descendente. Si no se especifica el sentido de ordenación, el orden ascendente es el valor predeterminado. Si sólo se especifica un campo (un nivel de ordenación) y está indexado, el índice se utiliza para la orden. Si el campo no está indexado o si hay más de un campo, el orden es secuencial.

Para varias ordenaciones (ordenaciones en varios campos), puede llamar a **ORDER BY ATTRIBUTE** tantas veces como sea necesario y especificar el parámetro opcional *, a excepción de la última llamada **ORDER BY ATTRIBUTE**, que inicia la operación de ordenación. Esta funcionalidad es útil para la gestión de ordenaciones múltiples en interfaces de usuario personalizadas. Tenga en cuenta que puede combinar las llamadas **ORDER BY ATTRIBUTE** con llamadas **ORDER BY**.

Nota: con esta sintaxis, puede pasar un solo nivel de ordenación (campo) por llamada **ORDER BY ATTRIBUTE**.

No importa qué forma de ordenación se haya definido, si la operación de ordenación va a tomar algún tiempo, 4D muestra automáticamente un mensaje que contiene un termómetro de progreso. Este mensaje se puede controlar mediante el uso de los comandos **MESSAGES ON** y **MESSAGES OFF**. Si se muestra el termómetro de progreso, el usuario puede hacer clic en el botón **Detener** para interrumpir la ordenación. Si se finaliza la ordenación, OK pasa a 1. De lo contrario, si se interrumpe la ordenación, OK pasa a 0 (cero).

Ejemplo

Usted desea ordenar la selección actual por edad (descendente) y luego por su nombre (ascendente). El orden por defecto es:

// [Customer]OB_Info contents partial export {"LastName":"Giorgio","age":33,"client":true}, {"LastName":"Sarah","age":42,"client":true}, {"LastName":"Mikken","age":"Forty-six","client":true}, {"LastName":"Wesson","age":44,"client":true}, {"LastName":"Johnson","age":44,"client":false}, {"LastName":"Hamp","age":"Sixty","client":true}, {"LastName":"Smeldorf","age":33,"client":true}, {"LastName":"Martin","client":true], {"LastName":"Evan","age":36,"client":true}, {"LastName":"Collins","age":33,"client":true,"Sex":"female"}, {"LastName":"Garbando","age":60,"client":false,"Sex":"male"}, {"LastName":"Smeldorf","age":54,"client":true}, {"LastName":"Smith","age":42,"client":true}, {"LastName":"Jones","age":52,"client":true}, {"LastName":"Kerrey","age":44,"client":true}, {"LastName":"Gordini","client":true}, {"LastName":"Delaferme","age":54,"client":true}, {"LastName":"Belami","age":"Forty-six","client":true}, {"LastName":"Smeldorf","age":22,"client":true}, {"LastName":"Smeldorf","age":70,"client":true}

Si ejecuta:

```
ORDER BY ATTRIBUTE([Customer];[Customer]OB_Info;"age";<;[Customer]OB_Info;"LastName";>)
```

Los registros están en el siguiente orden:

{"LastName":"Gordini","client":true}, //al inicio porque {"LastName":"Martin","client":true}, //age is null (missing) {"LastName":"Smeldorf","age":70,"client":true}
{"LastName":"Garbando","age":60,"client":false,"Sex":"male"}, {"LastName":"Delaferme","age":54,"client":true}, {"LastName":"Smeldorf","age":54,"client":true},
{"LastName":"Jones","age":52,"client":true}, {"LastName":"Johnson","age":44,"client":false}, {"LastName":"Kerrey","age":44,"client":true},
{"LastName":"Wesson","age":44,"client":true}, {"LastName":"Sarah","age":42,"client":true}, {"LastName":"Smith","age":42,"client":true},
{"LastName":"Evan","age":36,"client":true}, {"LastName":"Collins","age":33,"client":true,"Sex":"female"}, {"LastName":"Giorgio","age":33,"client":true},
{"LastName":"Smeldorf","age":33,"client":true}, {"LastName":"Smeldorf","age":22,"client":true}, {"LastName":"Hamp","age":"Sixty","client":true}, //string values in
age {"LastName":"Belami","age":"Forty-six","client":true}, //are handled separately {"LastName":"Mikken","age":"Forty-six","client":true}

ORDER BY FORMULA

ORDER BY FORMULA (tabla ; formula {; > o <}{; formula2 ; > o <2 ; ... ; formulaN ; > o <N})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual ordenar la selección de registros
formula	Expresión	➡	Fórmula de ordenación de los registros (puede ser de tipo Alfanumérico, Real, Entero, Entero largo, Fecha, Hora o Booleano)
> o <	Operador	➡	Sentido de la ordenación para cada nivel: > orden cresciente, u < orden decreciente

Descripción

ORDER BY FORMULA ordena (reordena) los registros de la selección actual de *tabla* para el proceso actual. Una vez efectuada la ordenación, el primer registro de la nueva selección actual se convierte en el nuevo registro actual.

Note que debe especificar *tabla*. No puede utilizar una tabla por defecto.

Puede ordenar la selección en uno o varios niveles. Para cada nivel de ordenación, usted pasa una expresión en *formula* y un criterio de ordenación en *> o <*. Si pasa el símbolo “mayor que” (>) el orden es creciente. Si pasa el simbolo "menor que " (<) el orden es decreciente. Si no especifica el criterio de ordenación, el orden por defecto es creciente.

El parámetro *formula* puede ser de tipo Alfa, Real, Entero, Entero largo, Fecha, Hora o Booleano.

Sin importar la manera en que se defina una ordenación, si la operación de ordenación va a tomar cierto tiempo, 4D muestra automáticamente un mensaje que contiene un termómetro de progreso. Estos mensajes pueden ser activados y desactivados utilizando los comandos **MESSAGES ON** y **MESSAGES OFF**. Si se muestra el termómetro de progresión, el usuario puede hacer clic en el botón Detener para interrumpir la ordenación. Si la ordenación se completa correctamente, OK toma el valor 1. De lo contrario, si la ordenación se interrumpe, OK toma el valor 0 (cero).

4D Server: este comando se ejecuta en el servidor, lo cual optimiza su ejecución. Note que cuando las variables son llamadas directamente en la *formula*, la ordenación se calcula con el valor de la variable en el equipo cliente. Por ejemplo, la instrucción **ORDER BY FORMULA([mytable];[mytable]myfield*myvariable)** se ejecutará en el servidor pero con el contenido de la variable myvariable del cliente..

Nota de compatibilidad: hasta 4D Server v11, este comando se ejecutaba en los equipos clientes. Por compatibilidad, este funcionamiento se conserva en las bases de datos convertidas. Sin embargo, una preferencia de compatibilidad y un selector del comando [#cmd id="642"/] permiten adoptar la ejecución en el servidor en bases de datos convertidas.

Ejemplo

Este ejemplo ordena los registros de la tabla [Personas] en orden descendente, basado en la longitud del apellido de cada persona. El registro de la persona con el apellido más largo será el primer registro de la selección actual:

```
ORDER BY FORMULA([Personas];Length([Personas]Apellido);<)
```

QUERY

QUERY ({tabla }{;}{ criterioBusqueda {; *}})

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla para la cual devolver una selección de registros o Tabla por defecto, si se omite
criterioBusqueda	Expresión	⇒	Criterio de búsqueda
*	Operador	⇒	Continuar la ejecución de la búsqueda

Descripción

El comando **QUERY** busca los registros que corresponden al criterio especificado en *criterioBusqueda* y devuelve una selección de registros de *tabla*. **QUERY** modifica la selección actual de *tabla* para el proceso actual y vuelve el primer registro de la nueva selección el registro actual.

Si omite el parámetro *tabla*, el comando se aplica a la tabla por defecto. Si no se ha definido una tabla por defecto, se genera un error.

Si no especifica *criterioBusqueda* ni el parámetro *, **QUERY** muestra la caja de diálogo del editor de búsquedas para *tabla* (excepto cuando es la última fila de una búsqueda múltiple, ver ejemplo 2):

Para mayor información sobre la utilización de este editor, consulte el Manual de Diseño.

El usuario construye la búsqueda, luego hace clic en el botón Buscar o Buscar en la selección. Si la búsqueda se realiza sin interrupciones, la variable sistema OK toma el valor 1. Si el usuario hace clic en Cancelar, el comando **QUERY** es interrumpido sin realizar la búsqueda y la variable OK toma el valor *0* (cero).

Ejemplo 1

El ejemplo siguiente muestra el editor de búsquedas para la tabla [Productos]:

```
QUERY([Productos])
```

Ejemplo 2

La línea siguiente muestra el editor de búsquedas para la tabla por defecto (si se ha definido)

```
QUERY
```

Si especifica el parámetro *criterioBusqueda*, el editor de búsquedas no se muestra y la búsqueda se define por programación. Para búsquedas simples (búsquedas en un solo campo) usted llama **QUERY** una vez con el parámetro *criterioBusqueda*. Para búsquedas complejas (búsquedas en múltiples campos o con múltiples condiciones), llame **QUERY** tantas veces como sea necesario con el parámetro *criterioBusqueda*, y el parámetro opcional *, excepto para la última llama **QUERY**, la cual inicia la búsqueda.

Ejemplo 3

El siguiente ejemplo busca las [Personas] cuyo nombre comienza por "a":

```
QUERY([Personas];[Personas]Nombre="a@")
```

Ejemplo 4

El siguiente ejemplo busca las [Personas] cuyo nombre comienza por "a" o "b":

```
QUERY([Personas];[Personas]Nombre="a@";*)  ` * indica que hay otro criterio de búsqueda
QUERY([Personas];l;[Personas]Nombre="b@")  ` Sin * indica el fin de la definición de los criterios de búsqueda y el inicio de la ejecución de la búsqueda.
```

Nota: el modo de interpretación del carácter @ en las búsquedas puede modificarse en una opción de las preferencias. Para mayor información, consulte la sección **Operadores de comparación**.

Construcción de una línea de búsqueda

- El parámetro *criterioBusqueda* utiliza la siguiente sintaxis:

{ operador; } campo comparador valor

- El operador se utiliza para unir las llamadas a **QUERY** cuando se definen búsquedas múltiples. Los operadores disponibles son los mismos del editor de búsquedas:

Operador	Símbolo a utilizar con QUERY
AND	&
OR	|
Except	#

El operador es opcional y no es necesario para la primera llamada a **QUERY** de una búsqueda múltiple, o si la búsqueda es una búsqueda simple.

- El *campo* es el campo a buscar. El *campo* puede pertenecer a otra tabla si pertenece a una tabla Uno relacionada a *tabla* con relación automática o manual. La tabla a la cual se aplica **QUERY** debe ser la tabla Muchos.
- El *operador* es el elemento que permite comparar *campo* y *criterioBusqueda*. Esta es la lista de posibles comparadores:

Comparador	Símbolo a utilizar con QUERY
Igual a	=
Diferente de	#
Menor que	<
Mayor que	>
Menor o igual a	<=
Mayor o igual a	>=

Nota: es posible definir el comparador bajo la forma de una expresión alfanumérica en lugar de un símbolo. En ese caso, es obligatorio utilizar punto y comas para disociar los elementos de la cadena de búsqueda. Este principio permite por ejemplo crear secuencias de búsquedas parámetrables variando el comparador, o construir interfaces de búsqueda usuario personalizadas. Consulte el ejemplo 21.

- El *valor* es el dato que se compara con el contenido de *campo*. El valor puede ser una expresión del mismo tipo que *campo*. El tipo de valor se evalúa una vez, al comienzo de la búsqueda y no para cada registro. Si la búsqueda se refiere al contenido de una cadena de caracteres, utilice en valor el símbolo arroba (@) para aislar el contenido a buscar, por ejemplo "@Perez@". Es de anotar, en este caso, que usted se beneficiará sólo de forma parcial de una búsqueda indexada (compactibidad de almacenamiento).
  La búsqueda por palabras claves está sólo disponible para campos tipo Alfa y Texto. Por favor consulte la sección **Operadores de comparación** para más información acerca de este tipo de búsqueda.

Estas son las reglas a tener en cuenta para la construcción de búsquedas múltiples:

- La primera línea no debe contener un operador.
- La siguientes líneas deben comenzar con un operador.
- Todas las líneas, excepto la última, deben utilizar el parámetro *.
- Para iniciar la búsqueda, no pase el parámetro * durante la construcción de su última línea. Alternativamente, puede ejecutar el comando **QUERY** sin otros parámetros diferentes a la tabla (el editor de búsquedas no se muestra; en su lugar, se ejecuta la búsqueda múltiple).

Nota: cada tabla mantiene su propia construcción de búsqueda actual. Esto significa que puede crear múltiples búsquedas simultáneamente, una por cada tabla. Debe utilizar el parámetro tabla o especificar una tabla por defecto.

Sin importar de qué manera se ha definido una búsqueda:

- Si la operación de búsqueda va a tomar algún tiempo, 4D muestra automáticamente un mensaje que contiene un termómetro de progreso. Estos mensajes pueden ser activados o desactivados utilizando los comandos **MESSAGES ON** y **MESSAGES OFF**. Si se muestra el termómetro de progreso, el usuario puede hacer clic en el botón Detener para interrumpir la búsqueda. Si la búsqueda se completa, OK toma el valor 1. De lo contrario, si la búsqueda es interrumpida, OK toma el valor 0 (cero).
- Si los campos indexados son especificados, la búsqueda es optimizada cada vez que sea posible (se busca primero en los campo indexados) reduciendo al máximo la duración de la operación. El comando usa los índices compuestos para las búsquedas utilizando **AND** (&)

Ejemplo 5

Buscamos los registros para que correspondan a personas con el apellido López:

```
QUERY([Personas];[Personas]Apellido="López")
```

Nota: si el campo Apellido está indexado, nos beneficiamos de una búsqueda acelerada utilizando el índice.

Recordatorio: esta búsqueda encontrará registros como "López", "lópez", "LÓPEZ", etc. Si quiere que la búsqueda tenga en cuenta las mayúsculas y minúsculas, defina criterios suplementarios que utilicen los códigos ASCII.

Ejemplo 6

El siguiente ejemplo busca los registros de personas llamadas Carlos López. El campo Apellido está indexado. El campo Nombre no está indexado.

```
QUERY([Personas];[Personas]Last Name="lópez";*)  ` Buscar todas las personas de apellido López
QUERY([Personas]; & ;[Personas]First Name="carlos")  ` llamadas Carlos
```

Cuando se realiza la búsqueda, primero se efectúa una búsqueda rápida en el campo indexado Apellido, y se reduce la selección de registros a las personas de apellido López. La búsqueda luego busca secuencialmente en el campo Nombre en esta selección

de registros.

Ejemplo 7

El siguiente ejemplo aprovechará automáticamente un índice compuesto de los campos *[People]First Name+[People]Last Name* (si existe) para encontrar los registros de todas las personas llamadas John Smith.

```
QUERY([People];[People]First Name="john";*)  ` Buscar a cada persona llamada John
QUERY([People];&;[People]Last Name="smith")  ` con apellido Smith
```

Para más información, consulte **Índices compuestos**.

Ejemplo 8

El siguiente ejemplo busca registros de personas de apellido López o Gómez. El campo Apellido está indexado.

```
QUERY([Personas];[Personas]Apellido="lópez";*)  ` Buscar todas las personas de apellido López···
QUERY([Personas];|;[Personas]Apellido="gómez")  ` ...o Gómez
```

El comando **QUERY** utiliza el índice del campo Apellido para ambas búsquedas. Las dos búsquedas se efectúan, y sus resultados se colocan en conjuntos internos que son combinados eventualmente utilizando una operación de unión.

Ejemplo 9

El siguiente ejemplo busca los registros de personas que no trabajan en una empresa. La búsqueda se realiza probando si el nombre de la empresa es una cadena vacía.

```
QUERY([Personas];[Personas]Empresa="")  ` Buscar las personas sin empresa
```

Ejemplo 10

El siguiente ejemplo busca cada persona cuyo apellido es López, y trabaja para una empresa en Barcelona. La segunda búsqueda utiliza un campo de otra tabla. Esta búsqueda se puede efectuar porque la tabla [Personas] está relacionada a la tabla [Empresa] por una relación muchos a uno:

```
QUERY([Personas];[Personas]Apellido="lópez";*)  ` Buscar todas las personas de apellido López···
QUERY([Personas];&;[Empresa]Ciudad ="Barcelona")  ` ... que trabajan para una empresa en Barcelona
```

Ejemplo 11

El siguiente ejemplo busca el registro de cada persona cuyo inicial del nombre esté entre la letra A (incluida) y M (incluida):

```
QUERY([Personas];[Personas]Nombre<"n")  ` Encontrar todas las personas entre A y M
```

Ejemplo 12

El siguiente ejemplo busca los registros de las personas que viven en Madrid o Barcelona:

```
QUERY([Personas];[Personas]CodigoPostal="28@";*)  ` Buscar toda las personas que viven en Madrid···
QUERY([Personas];|;[Personas]ZIP CodigoPostal="08@")  ` ...o en Barcelona
```

Ejemplo 13

Búsqueda por palabra clave: el siguiente ejemplo busca en toda la tabla [Productos] los registros cuyo campo Descripción contenga la palabra "fácil":

```
QUERY([Productos];[Productos]Descripcion="fácil")  ` Buscar productos cuya descripción contenga la palabra clave fácil
```

Ejemplo 14

El siguiente ejemplo busca los registros que corresponden a la referencia de la factura introducida en una caja de diálogo:

```
vBuscar:=Request("Introducir una referencia de factura:")  ` Obtener una referencia de factura del usuario
If(OK=1)  ` Si el usuario hace clic en OK
   QUERY([Factura];[Factura]Ref=vBuscar)  ` Buscar la referencia de factura que corresponda a vBuscar
End if
```

Ejemplo 15

El siguiente ejemplo busca los registros de facturas introducidas en 1996. Buscamos todos los registros introducidos entre el 31/12/95 y 1/1/97:

```
QUERY([Facturas];[Facturas]FechaFactura >!31/12/95!;*)  ` Buscar facturas después de 31/12/95···
QUERY([Facturas];&;[Facturas]FechaFactura <!1/1/97!)  ` y antes de 1/1/97
```

Ejemplo 16

El siguiente ejemplo busca los empleados cuyo salario está entre $10 000 y $50 000. La búsqueda incluye los empleados que ganan $10 000, pero excluye a los que ganan $50 000:

```
QUERY([Empleados];[Empleados]Salario >=10000;*)  ` Buscar los empleados que tengan un salario entre···
QUERY([Empleados];&;[Empleados]Salario <50000)  ` ...$10 000 y $50 000
```

Ejemplo 17

El siguiente ejemplo busca los empleados del departamento de mercadeo con salarios superiores a $20 000. Se busca primero en el campo Salario porque está indexado. Observe que la segunda búsqueda utiliza un campo de otra tabla. Esto es posible porque la tabla [Dept] está relacionada a la tabla [Empleados] por una relación automática de muchos a uno. Aunque el campo [Dept]Nombre está indexado, la búsqueda no es indexada porque la relación debe ser activada secuencialmente para cada registro en la tabla [Empleados]:

```
QUERY([Empleados];[Empleados]Salario >20000;*)  ` Buscar los empleados con salarios superiores a $20 000 y...
QUERY([Empleados];&;[Dept]Nombre="mercadeo")  ` ...que trabajen en el departamento de mercadeo
```

Ejemplo 18

Se tienen tres tablas relacionadas de muchos a uno: [Ciudad] -> [Departamento] -> [Region]. El siguiente ejemplo busca las regiones cuyas ciudades comienzan con "New".

```
QUERY([Region];[Ciudad]Nombre="New")  ` Buscar todas las regiosnes cuyas ciudades comienzan por "New"
```

Ejemplo 19

El siguiente ejemplo busca la información igual al valor de la variable *miVar*.

```
QUERY([Leyes];[Leyes]Texto =miVar)  ` Buscar todas las leyes que son iguales al valor de miVar
```

La búsqueda puede tener muchos resultados diferentes, dependiendo del valor de *miVar*. La búsqueda se realizará también de manera diferente. Por ejemplo:

- Si *miVar* es igual a *"Copyright@"*, la selección contiene todas las leyes que contienen textos que comienzan por Copyright.
- Si *miVar* es igual a *"@Copyright@"*, la selección contiene todas las leyes que contienen al menos una ocurrencia de Copyright.

Ejemplo 20

El siguiente ejemplo añade o no líneas a una búsqueda compleja dependiendo del valor de las variables. De esta forma, sólo los criterios válidos son tenidos en cuenta para la búsqueda.

```
QUERY([Factura];[Factura]Pagada=False;*)
If($ciudad#"")  ` si se ha especificado un nombre de ciudad `
   QUERY([Factura];[Factura]Ciudad_entrega=$ciudad;*)
End if

If($Codigo_Postal#"")  ` si se ha especificado un código postal

   QUERY([Factura];[Factura]Codigo_Postal=$Codigo_Postal;*)
End if
QUERY([Factura])  ` Ejecución de la búsqueda sobre los criterios
```

Ejemplo 21

Este ejemplo ilustra la utilización de un operador de comparación como expresión alfanumérica. El valor del operador de comparación está definido a través de un menú desplegable ubicado en una caja de diálogo de búsqueda personalizada:

```
C_TEXT($oper)
$oper:=_popup_operator{_popup_operator}  `$oper igual por ejemplo "#" o "="
```

```
If(OK=1)
   QUERY([Factura];[Factura]Cantidad;$oper;$cantidad)
End if
```

Ejemplo 22

El uso de los índices de palabras claves puede acelerar de manera importante sus aplicaciones.

```
QUERY([IMAGENES];[IMAGENES]Fotos%"gatos")  // buscar fotos asociadas con la palabra clave "gatos"
```

Variables y conjuntos del sistema

Si la búsqueda se lleva a cabo correctamente, la variable sistema OK toma el valor 1.
La variable OK toma el valor 0 si: - el usuario hace clic en Cancelar en la caja de diálogo de búsqueda,
- en modo 'búsqueda y bloqueo' (ver el comando **SET QUERY AND LOCK**), la búsqueda encuentra al menos un registro bloqueado. En este caso igualmente, el conjunto sistema LockedSet se actualiza.

QUERY BY ATTRIBUTE

QUERY BY ATTRIBUTE ({tabla}{;}{opConj ;} campoObjeto ; rutaAtributo ; opBusq ; valor {; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual devolver una selección de registros o Tabla por defecto si se omite
opConj	Operador	➡	Operador de conjunción ausar para combinar varias búsquedas (si las hay)
campoObjeto	Campo	➡	Campo objeto cuyos atributos utilizar para la búsqueda
rutaAtributo	Cadena	➡	Nombre o ruta de atributo
opBusq	Cadena, Operador	➡	Operador de búsqueda (comparador)
valor	Texto, Número, Fecha, Hora	➡	Valor a comparar
*	Operador	➡	Espera de ejecución de la búsqueda

Descripción

QUERY BY ATTRIBUTE busca los registros que coincidan con la cadena de consulta definida utilizando los parámetros *campoObjeto*, *rutaAtributo*, *opBusq* y *valor*, y devuelve una selección de registros para *tabla*.

QUERY BY ATTRIBUTE cambia la selección actual de *tabla* para el proceso actual y vuelve el primer registro de la nueva selección el registro actual. Si se omite el parámetro *tabla*, el comando se aplica a la tabla por defecto. Si no se ha definido ninguna tabla por defecto, se produce un error.

El parámetro opcional *opConj* se utilizar para combinar varias llamadas a **QUERY BY ATTRIBUTE** en caso de búsquedas múltiples. Los operadores de conjunción disponibles son los mismos que los del comando **QUERY**:

Conjunción	Símbolo a utilizar con QUERY BY ATTRIBUTE
AND	&
OR	|
Except	#

El parámetro *opConj* no se utiliza para la primera llamada a **QUERY BY ATTRIBUTE** de una búsqueda múltiple, o si la búsqueda es una búsqueda simple. Si lo omite dentro de una búsqueda múltiple, el operador AND (&) se utiliza por defecto.

En *campoObjeto*, pase el campo objeto cuyos atributos desea buscar. Si pertenece a una tabla Uno relacionada a *tabla* con una relación automática o manual, el *campoObjeto* pueden pertenecer a otra tabla.
QUERY BY ATTRIBUTE soporta los atributos personalizados 4D Write Pro cuando los documentos son almacenados en campos Objeto. Para mayor información, consulte la sección **Almacenar los documentos 4D Write Pro en los campos objeto 4D**.

En *rutaAtributo*, pase la ruta del atributo cuyos valores desea comparar para cada registro, por ejemplo "niños.niñas.edad". Si pasa un simple nombres, por ejemplo "lugar", designa el atributo correspondiente ubicado en el primer nivel del campo objeto. Si un atributo "x" es un array, **QUERY BY ATTRIBUTE** buscará registros que contengan un atributo "x" en el cual al menos un elemento coincida con los criterios. Para buscar entre los atributos de array, es necesario indicar al comando **QUERY BY ATTRIBUTE** que el atributo "x" es un array añadiendo ".[]" a su nombre en el parámetro *rutaAtributo* (ver ejemplo 3). Puede añadir una letra entre brackets (i.e. "[b]") para enlazar los argumentos (ver el párrafo **Asociar los criterios para las búsquedas en los elementos de array** abajo).

Notas:

- Tenga en cuenta que los nombres de atributos tienen en cuenta las mayúsculas y minúsculas: puede tener diferentes nombres de atributos "MyAtt" y "myAtt" en el mismo registro.
- Los nombres de atributos se recortan para eliminar espacios adicionales. Por ejemplo, "mi primer atributo .mi segundo atributo" se interpreta como "mi primer atributo.mi segundo atributo".

El parámetro *opBusq* es el operador de comparación que se aplica entre *campoObjeto* y *valor*. Puede pasar uno de los símbolos que se muestran aquí:

Comparación	Símbolo a utilizar con QUERY BY ATTRIBUTE
Igual a	=
Diferente de (*)	#
Menor que	<
Mayor que	>
Menor o igual a	<=
Mayor o igual a	>=

(*) Cuando se utiliza con los elementos del array, el operador # significa "no contiene ninguno".

Nota: se puede especificar el operador de comparación como una expresión texto en lugar de un símbolo. Consulte la descripción del comando **QUERY** para más información.

valor valor es el dato contra el que se va a comparar *rutaAtributo*. El valor que puede ser cualquier expresión del mismo tipo que *rutaAtributo*. El valor se evalúa una vez, al inicio de la búsqueda. El valor no se evalúa para cada registro. Para buscar una cadena dentro de una cadena (una búsqueda "contains"), utilice el símbolo arroba (@) en *valor* para aislar la cadena a buscar, como se muestra en este ejemplo: "@Smith@". Note que en este caso, la búsqueda sólo se beneficia parcialmente desde el índice (compacidad de almacenamiento de datos).

Esta es la estructura de una consulta por atributos:

```
QUERY BY ATTRIBUTE([Table] ;[Table]ObjectField ;"attribute1.attribute2";=;value)
```

Nota: un criterio implícito para todos los operadores (excepto #) es que el campo Objeto contiene un atributo. Sin embargo, para el operador #, puede definirse (ver más adelante).

Uso del operador

Por ejemplo, la siguiente búsqueda devuelve los registros de las personas que tienen un perro cuyo nombre no es Rex (y no los registros de las personas que no tienen un perro, o que tienen un perro sin nombre):

Las búsquedas por atributo que utilizan el operador "#" pueden tener resultados diferentes dependiendo de si la propiedad está seleccionada para el campo objeto:

- Propiedad **Traducir los NULL en valores vacíos** seleccionada (valor por defecto, recomendado en la mayoría de los casos).
  En este caso, el operador "#" debe ser visto como seleccionando los registros donde "ningún atributo" del campo contiene el valor buscado. En este contexto, 4D considera de manera similar:
  - los campos para las cuales el valor del atributo es diferente al valor de la búsqueda,
  - los campos donde el atributo no está presente (o contiene un valor Null).

  Por ejemplo: la siguiente búsqueda devuelve los registros de personas que tienen un perro cuyo nombre no es Rex, así como también los registros de personas que no tienen perro, o que tienen perro si nombre:

  ```
  QUERY BY ATTRIBUTE([People];[People]Animals;"dog.name";#;"Rex")
  ```

  Otro ejemplo: esta búsqueda devolverá todos los registros para los que [Table]*ObjectField* contiene un objeto que contiene un atributo *attribute1* que es en sí un objeto que contiene un atributo *attribute2* cuyo valor no es el valor (no va a devolver los objetos que no contienen *attribute1* o *attribute2*):

  ```
  QUERY BY ATTRIBUTE([Table];[Table]ObjectField;"attribute1.attribute2";#;value)
  ```

  Este principio también aplica a los atributos array. Por ejemplo:

  ```
  QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].city";#;"paris")
  ```

  Esta búsqueda devolverá los registros de las personas que no tienen una dirección en París.

  Para obtener específicamente los registros donde el atributo no está definido, puede utilizar un objeto vacío (ver ejemplo 2). Note sin embargo que la búsqueda de valores NULL en elementos array no es soportada.

- Propiedad **Traducir los NULL a valores vacíos** no seleccionada (modo "SQL").
  En este caso, los atributos no definidos (atributos no presentes en el campo o cuyo valor es Null) no se consideran como equivalentes a los valores vacíos por defecto. Como resultado, las búsquedas del tipo "atributo A es diferente del atributo B" no devolverán registros en los que el atributo A no está definido.
  Para utilizar el mismo ejemplo anterior, cuando la opción **Traducir los NULL en valores vacíos** no se selecciona para el campo [People]Animals, la siguiente búsqueda sólo devolverá registros para las personas que tienen un perro cuyo atributo "name" no contiene " Rex ". Los registros de las personas que no tienen perro, o que tienen un perro sin nombre no se devolverán en este caso.

  ```
  QUERY BY ATTRIBUTE([People];[People]Animals;"dog.name";#;"Rex")
  ```

  Esta operación, más cerca de la lógica SQL, se reserva para necesidades específicas.

Crear búsquedas múltiples

Aquí están las reglas a seguir para la construcción de varias búsquedas por atributo:

- La primera línea no debe contener una conjunción.
- Cada argumento de búsqueda sucesivo puede comenzar con una conjunción. Si lo omite, el operador AND (&) se utiliza por defecto.
- Todas las líneas, excepto la última, deben utilizar el parámetro *.
- **QUERY BY ATTRIBUTE** se puede combinar con los comandos **QUERY** (ver ejemplo).
- Para realizar la búsqueda, no especifique el parámetro * en el último comando **QUERY BY ATTRIBUTE**. Alternativamente, puede ejecutar el comando **QUERY** sin parámetros distintos a la tabla.

Nota: cada tabla mantiene su propia construcción de búsqueda actual. Esto significa que puede crear varias búsquedas simultáneamente, una para cada tabla.

No importa la forma en que una búsqueda se haya definido:

- Si la operación de búsqueda va a tomar algún tiempo para llevarse a cabo, 4D muestra automáticamente un mensaje que contiene un termómetro de progreso. Estos mensajes se pueden activar y desactivar mediante el uso de los comandos **MESSAGES ON** y **MESSAGES OFF**. Si se muestra un termómetro de progreso, el usuario puede hacer clic en el botón Detener para interrumpir la búsqueda. Si se completa la consulta, OK toma el valor 1. De lo contrario, si la consulta se interrumpe, OK toma el valor 0 (cero).
- Si no se especifica ningún campo objeto indexado, la búsqueda se optimiza cada vez que es posible (los campos indexados se buscan primero), resultando en una búsqueda que toma la menor cantidad de tiempo posible.

Valores fecha en el objeto

Las fechas se almacenan en los objetos en función de los parámetros de la base; por defecto, se tiene en cuenta la zona horaria (ver el selector JSON use local time en el comando **SET DATABASE PARAMETER**).

```
!1973-05-22! -> "1973-05-21T23:00:00.000Z"
```

Este ajuste también se tiene en cuenta durante las búsquedas, por lo que no tiene que preocuparse por ello si siempre utiliza su base en el mismo lugar y si los parámetros son los mismos en todos los equipos que acceden a los datos. En este caso, la siguiente búsqueda devolverá correctamente los registros cuyo atributo Birthday sea igual a !1973-05-22! (guardada como "1973-05-21T23:00:00.00Z"):

```
QUERY BY ATTRIBUTE([Persons];[Persons]OB_Info;"Birthday";=;!1973-05-22!)
```

Si no desea utilizar el parámetro GMT, puede modificar estos parámetros utilizando la siguiente instrucción:

```
SET DATABASE PARAMETER(JSON use local time;0)
```

Tenga en cuenta que el alcance de este parámetro está limitado al process. Si ejecuta esta instrucción, el 1 de octubre de 1965 se almacenará "1965-10-01T00: 00: 00.000Z" pero usted deberá ajustar el mismo parámetro antes de lanzar sus búsquedas:

```
SET DATABASE PARAMETER(JSON use local time;0)
QUERY BY ATTRIBUTE([Persons];[Persons]OB_Info;"Birthday";=;!1976-11-27!)
```

Utilización de la propiedad virtual longitud

Puede utilizar la propiedad virtual "longitud" con este comando. Esta propiedad está disponible automáticamente para todos los atributos de tipo array y devuelve el tamaño del array, es decir, el número de elementos que contiene. Se puede utilizar en el contexto de la ejecución del comando **QUERY BY ATTRIBUTE** (ver ejemplo 4).

Asociar los criterios para las búsquedas en los elementos de array

(Nuevo en 4D v16 R2) Al buscar en atributos array con varios argumentos de búsqueda unidos por el operador AND, puede que quiera asegurarse de que sólo se devuelvan los registros que contengan elementos que coincidan con todos los argumentos y no los registros donde se pueden encontrar argumentos en diferentes elementos. Para ello, debe vincular los argumentos de búsqueda a elementos del array, de modo que solo se encuentren elementos únicos que contengan argumentos vinculados.

Por ejemplo, con los dos registros siguientes:

```
{ "name":"martin", "locations" : [ { "kind":"home", "city":"paris" } ] } , { "name":"smith", "locations" : [ { "kind":"home", "city":"lyon" } , { "kind":"office", "city":"paris" } ] }
```

Usted quiere encontrar la gente con un tipo de ubicación "home" en la ciudad "Paris". Si escribe:

```
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].city";=;"paris";*)
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].kind";=;"home")
```

... la búsqueda devolverá "martin" y "smith" porque "smith" tiene un elemento "locations" cuyo "kind" es "home" y un elemento "locations" cuya "city" es "paris", aunque son elementos diferentes.

Si sólo desea obtener los registros donde los argumentos coincidentes estén en el mismo elemento, debe **asociar los criterios**. Para vincular criterios de búsqueda:

- Agregue una letra entre el [] de la primera ruta a asociar y repita la misma letra en todos los argumentos vinculados. Por ejemplo: **locations[a].city** y **locations[a].kind**. Puede utilizar cualquier letra del alfabeto latino (no sensible a mayúsculas y minúsculas).
- Para agregar diferentes criterios vinculados en la misma búsqueda, utilice otra letra (vea los ejemplos a continuación). Puede crear hasta 26 combinaciones de criterios en una sola búsqueda.

Con los registros anteriores, si escribe:

```
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[a].city";=;"paris";*)
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[a].kind";=;"home")
```

... la búsqueda sólo devolverá "martin" porque tiene un elemento "locations" cuyo "kind" es "home" y cuya "city" es "paris". La consulta no devolverá "smith" porque los valores "home" y "paris" no están en el mismo elemento de array. Vea los ejemplos a continuación para ver más ejemplos de esta funcionalidad.

Nota: utilizar la sintaxis relacionada en una sola línea de búsqueda dará los mismos resultados que una búsqueda estándar, excepto cuando se utiliza el operador "#": en este caso, se pueden devolver resultados no válidos. Por lo tanto, esta sintaxis específica no es soportada.

Ejemplo 1

En este ejemplo, el atributo "age" es una cadena o un entero y queremos encontrar personas cuyas edades estén entre 20 y 29. Las primeras dos líneas buscan el atributo como un entero (>=20 y <30) y las últimas consultan el campo como una cadena (comienza por "2" pero es diferente de "2".)

```
QUERY BY ATTRIBUTE([Persons];[Persons]OB_Info;"age";>=;20;*)
QUERY BY ATTRIBUTE([Persons]; & ;[Persons]OB_Info;"age";<;30;*)
QUERY BY ATTRIBUTE([Persons];I;[Persons]OB_Info;"age";=;"2@";*)
QUERY BY ATTRIBUTE([Persons]; & ;[Persons]OB_Info;"age";#;"2") //sin * para lanzar la ejecución
```

Ejemplo 2

El comando **QUERY BY ATTRIBUTE** se puede utilizar para encontrar registros en los que algunos atributos se definen (o no). Para ello, debe utilizar un objeto vacío.

```
//Buscar los registros donde el correo electrónico se define en el campo objeto
C_OBJECT($undefined)
QUERY BY ATTRIBUTE([Persons];[Persons]Info;"email";#;$undefined)
```

```
//Buscar los registros donde el código postal no está definido en el campo objeto
C_OBJECT($undefined)
QUERY BY ATTRIBUTE([Persons];[Persons]Info;"zip code";=;$undefined)
```

Ejemplo 3

Usted quiere buscar un campo que contiene los atributos array. Con los dos registros siguientes:

```
{ "name":"martin", "locations" : [ { "kind":"office", "city":"paris" } ] } , { "name":"smith", "locations" : [ { "kind":"home", "city":"lyon" } , { "kind":"office", "city":"paris" } ] }
```

... **QUERY BY ATTRIBUTE** encontrará personas con una ubicación en "paris" utilizando esta instrucción:

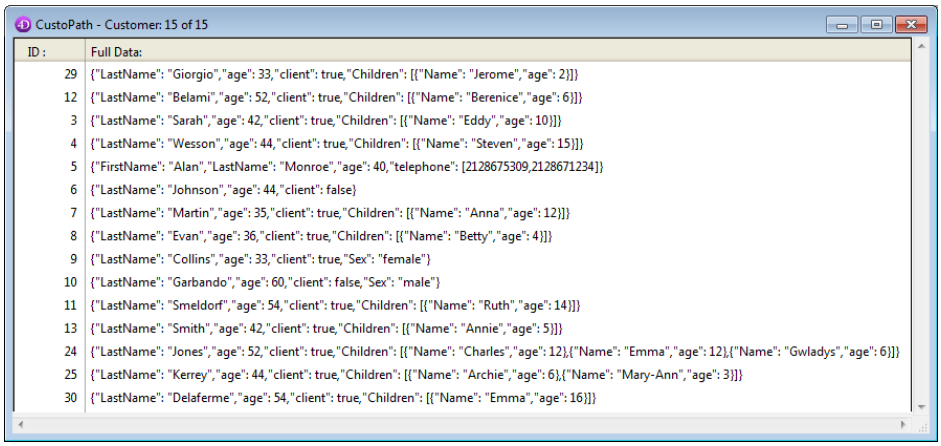
```
//indica el atributo array con la sintaxis "[]"
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].city";=;"paris")
//Selecciona "martin" y "smith"
```

Nota: si ha definido varios criterios en el mismo atributo array, los criterios coincidentes no se aplicarán necesariamente al mismo elemento de array. En el siguiente ejemplo, la búsqueda devolverá "smith" porque tiene un elemento "locations" cuyo "kind" es "home" y un elemento "locations" cuya "city" es "paris", Incluso si no es el mismo elemento:

```
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].kind";=;"home";*)
QUERY BY ATTRIBUTE([People]; & ;[People]OB_Field;"locations[].city";=;"paris")
//Selecciona "smith"
```

Ejemplo 4

Este ejemplo ilustra el uso de la propiedad "longitud" virtual. Su base tiene un campo objeto [Customer]full_Data con los siguientes datos:



Usted quiere obtener los registros de los clientes que tienen dos o más hijos. Para ello, se puede escribir:

```
QUERY BY ATTRIBUTE([Customer];[Customer]full_Data;"Children.length";>=;2)
```

Ejemplo 5

Estos ejemplos ilustran las diversas combinaciones disponibles de argumentos de consulta vinculados en arrays. Asumiendo que tiene los siguientes registros:

Record1:
[Person]Name: "Sam"
[Person]ObjectField:

```
  "Children": [ {
     "Name": "Harry",
     "Age": "15",
     "Toy": [ {
        "Name": "Car",
        "Color": "Blue"
      }, {
        "Name": "Teddy Bear",
        "Color": "Brown"
     } ]
    }, {
     "Name": "Betty",
     "Age": "9",
     "Toy": [ {
        "Name": "Car",
```

```
        "Color": "Green"
      }, {
        "Name": "Puzzle",
        "Color": "Blue"
      } ]
    } ]
```

Record2:

```
[Person]Name: "Louis"
[Person]ObjectField:
   "Children": [ {
      "Name": "Harry",
      "Age": "15",
      "Toy": [ {
        "Name": "Water gun",
        "Color": "Blue"
      } ]
    }, {
      "Name": "Betty",
      "Age": "3",
      "Toy": [ {
        "Name": "Car",
        "Color": "Blue"
      }, {
        "Name": "Puzzle",
        "Color": "Green"
      } ]
    } ]
```

Record3:

```
[Person]Name: "Victor"
[Person]ObjectField:
   "Children": [ {
      "Name": "Harry",
      "Age": "9",
      "Toy": [ {
        "Name": "Doll",
        "Color": "Pink"
      }, {
        "Name": "Puzzle",
        "Color": "Blue"
      } ]
    }, {
      "Name": "Betty",
      "Age": "15",
      "Toy": [ {
        "Name": "Water gun",
        "Color": "Blue"
      } ]
    } ]
```

Para encontrar personas que tienen un hijo llamado "Betty" que tiene 15 años:

```
QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[a].Name";=;"Betty";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[a].Age";=;"15")
 //returns "Victor"

QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[].Name";=;"Betty";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[].Age";=;"15")
 //returns "Sam", "Louis" and "Victor"
```

Para encontrar personas que tienen un hijo llamado "Betty" que tiene 15 años y un hijo llamado "Harry" que tiene 9 años:

```
QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[a].Name";=;"Betty";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[a].Age";=;"15";*)
QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[b].Name";=;"Harry";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[b].Age";=;"9")
 //returns "Victor"

QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[].Name";=;"Betty";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[].Age";=;"15";*)
QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[].Name";=;"Harry";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[].Age";=;"9")
 //devuelve"Sam" y "Victor"
```

Para buscar personas que tienen un hijo de 15 años llamado "Harry" con un juguete "blue car" (buscar en un array de arrays):

```
QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[a].Name";=;"Harry";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[a].Age";=;"15";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[a].Toy[b].Name";=;"Car";*)
```

```
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[a].Toy[b].Color";=;"Blue")
 //returns "Sam"

QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[].Name";=;"Harry";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[].Age";=;"15";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[].Toy[].Name";=;"Car";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[].Toy[].Color";=;"Blue")
 //devuelve "Sam" y "Louis"
```

Variables y conjuntos del sistema

Si la búsqueda se lleva a cabo correctamente, la variable sistema OK toma el valor 1.
La variable OK toma el valor 0 si: - el usuario hace clic en Cancelar en la caja de diálogo de búsqueda,
- en modo 'búsqueda y bloqueo' (ver el comando **SET QUERY AND LOCK**), la búsqueda encuentra al menos un registro bloqueado. En este caso igualmente, el conjunto sistema LockedSet se actualiza.

QUERY BY EXAMPLE

QUERY BY EXAMPLE ({tabla}{;}{*})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual debe devolverse una selección de registros , o Tabla por defecto, si se omite
*	Operador	➡	Si se pasa, no se muestra la barra de desplazamiento

Descripción

QUERY BY EXAMPLE efectúa la misma acción que el comando de menú Búsqueda por formulario... en el entorno Diseño. Este comando muestra el formulario de entrada actual como ventana de búsqueda. **QUERY BY EXAMPLE** busca en *tabla* los datos que el usuario introduce en la ventana de búsqueda. El formulario debe contener los campos que usted quiere utilizar para efectuar la búsqueda. La búsqueda se optimiza; los campos indexados se utilizan automáticamente para optimizar la búsqueda.

Ver el manual de Diseño de 4D para mayor información sobre la utilización del comando de menú Búsqueda por formulario... del entorno Diseño.

Ejemplo

El método en este ejemplo muestra el formulario miBusqueda. Si el usuario acepta el formulario y ejecuta la búsqueda (es decir, si la variable sistema OK toma el valor 1), se muestran los registros que cumplen con el criterio de búsqueda:

```
FORM SET INPUT([Personas];"miBusqueda") ` Pasar al formulario de entrada
QUERY BY EXAMPLE([Personas]) ` Mostrar el formulario y realizar la búsqueda
If(OK=1) ` Si el usuario valida la búsqueda
   DISPLAY SELECTION([Personas]) ` Mostrar los registros
End if
```

Variables y conjuntos del sistema

Si el usuario hace clic en el botón Aceptar o presionan la tecla Enter, la variable sistema OK toma el valor 1 y la búsqueda se realiza. Si el usuario hace clic en el botón Cancelar o presiona la tecla de anulación, la variable sistema OK toma el valor 0 y la búsqueda se cancela.

QUERY BY FORMULA

QUERY BY FORMULA (tabla {; formula})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla en la cual efectuar la búsqueda
formula	Booleano	➡	Fórmula de búsqueda

Descripción

QUERY BY FORMULA busca registros en *tabla*. **QUERY BY FORMULA** modifica la selección actual de *tabla* para el proceso actual y hace del primer registro el nuevo registro actual.

QUERY BY FORMULA y **QUERY SELECTION BY FORMULA** funcionan exactamente de la misma manera, excepto que **QUERY BY FORMULA** busca en todos los registros de la tabla y **QUERY SELECTION BY FORMULA** busca únicamente en los registros de la selección actual.

Los dos comandos aplican *formula* a cada registro de la tabla o de la selección. *formula* es una expresión booleana que debe devolver TRUE o FALSE. Si *formula* devuelve TRUE, el registro está incluido en la nueva selección.

El parámetro *formula* puede ser simple, como la comparación de un campo con un valor; o compleja, como la realización de un calculo o incluso una evaluación de los valores en una tabla relacionada. *formula* puede ser una función 4D (comando), o una expresión que usted haya creado. Cuando trabaje con campos de tipo Alfa o Texto, puede utilizar en *formula* símbolos arroba (@) como también el operador "contiene" (%) para búsquedas de palabras claves. Para mayor información, por favor consulte la descripción del comando **QUERY.**

Si omite el parámetro *formula*, 4D muestra la caja de diálogo de búsqueda (el usuario puede añadir una línea de fórmula efectuando **Alt+clic** en el botón **[+]**).

Cuando termina la búsqueda, el primer registro de la nueva selección es cargado desde el disco y se convierte en el registro actual.

Estos comandos son optimizados y pueden particularmente aprovechar los índices. Cuando el tipo de búsqueda lo permite, estos comandos ejecutan búsquedas equivalentes al comando **QUERY.** Por ejemplo, la instrucción QUERY BY FORMULA ([mitabla]; [mitabla]micampo=valor) se ejecutará igual que QUERY ([mitabla]; [mitabla]micampo=valor), que permite utilizar índices. 4D también puede optimizar búsquedas que contengan partes que no puedan ser optimizadas, ejecutando primero las partes optimizables y luego combinando los resultados con el resto de la búsqueda. Por ejemplo, la instrucción QUERY BY FORMULA ([mitabla]; Length(micampo)=valor1 | micampo=valor2) se optimizará parcialmente.

Estos comandos por defecto efectúan "uniones" como SQL cuando compara campos de diferentes tablas. Esto significa que no es necesario que exista una relación automática estructural entre las tablas. Por ejemplo, puede ejecutar una instrucción del tipo QUERY BY FORMULA([Table_A];([Table_A]field_X = [Table_B]field_Y) & ([Table_B]field_Y = "abc")) (ver ejemplo 3). La primera parte de la fórmula ([Table_A]field_X = [Table_B]field_Y) establece la unión entre dos campos y la segunda parte ([Table_B]field_Y = "abc") define el criterio de búsqueda (al menos un criterio debe definirse).
Si existen, las relaciones entre tablas, en principio no se utilizan. Sin embargo, estos comandos utilizarán relaciones automáticas en los siguientes casos:

- Si la formula no puede descomponerse en elementos de la forma { campo ; comparador ; valor}
- Si dos campos de la misma tabla son comparados.

Nota: por razones de compatibilidad, es posible desactivar el mecanismo de uniones, bien sea globalmente vía las Preferencias de la base (bases de datos convertidas únicamente) o por procesos utilizando el comando **SET DATABASE PARAMETER.**

4D Server: este comando se ejecuta en el servidor, lo cual optimiza su ejecución. Note que cuando las variables son llamadas directamente en *formula*, la ordenación se calcula con el valor de la variable en el equipo cliente. Por ejemplo, la instrucción **QUERY BY FORMULA** ([mitabla]; [mitabla]micampo=mivariable) se ejecutará en el servidor pero con el contenido de la variable mivariable del equipo cliente.

Compatibility note: hasta 4D Server v11, este comando se ejecutaba en el equipo cliente. Por compatibilidad, este funcionamiento se conserva en las bases de datos convertidas. Sin embargo, una propiedad de compatibilidad y un selector del comando **SET DATABASE PARAMETER** permiten adoptar la ejecución en el servidor en bases de datos convertidas.

Ejemplo 1

Este ejemplo busca los registros para todas las facturas que se introdujeron en diciembre de cualquier año. El principio consiste en aplicar la función **Month of** a cada registro. Esta búsqueda no podría realizarse de otra forma sin crear un campo separado para el mes:

```
QUERY BY FORMULA([Facturas];Month of([Facturas]Entrada)=12)  ` Buscar las facturas entradas en diciembre
```

Ejemplo 2

Este ejemplo busca los registros de las personas que tienen nombres con más de 10 caracteres:

```
QUERY BY FORMULA([Personas];Length([Personas]Nombre)>10)  ` Buscar nombres de más de diez caracteres
```

Ejemplo 3

Este ejemplo activa las uniones SQL para una búsqueda por fórmula específica:

```
$valorActual:=Get database parameter(QUERY BY FORMULA Joins)
SET DATABASE PARAMETER(QUERY BY FORMULA Joins;2)  `Activar uniones SQL
  `Buscar todas la líneas de facturas del cliente "ACME" aunque las tablas no estén relacionadas
QUERY BY FORMULA([Linea_Facturas];([Linea_Facturas]Id_Factura=[Facturas]Id &[Facturas]Cliente="ACME"))
SET DATABASE PARAMETER(QUERY BY FORMULA Joins;$valorActual)  `Se reestablece la configuración actual
```

QUERY SELECTION

QUERY SELECTION ({tabla }{;}{ criterioBusqueda {; *}})

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla en la cual efectuar la búsqueda o Tabla por defecto, si se omite
criterioBusqueda	Expresión	⇒	Líneas de búsqueda
*	Operador	⇒	Bandera para continuar la búsqueda

Descripción

QUERY SELECTION busca registros en *tabla*.El comando **QUERY SELECTION** cambia la selección actual de *tabla* para el proceso actual y hace que el primer registro de la nueva selección sea el registro actual.

QUERY SELECTION funciona y realiza las mismas acciones que **QUERY**. La diferencia entre los dos comandos es el alcance de la búsqueda:

- **QUERY** busca registros entre los registros en la tabla.
- **QUERY SELECTION** busca registros entre los registros de la selección actual de la tabla.

Para mayor información, consulte la descripción del comando **QUERY**.

El comando **QUERY SELECTION** es útil cuando una búsqueda no se puede definir utilizando una secuencia de llamadas enlazadas **QUERY** con el parámetro *. Por lo general, es el caso cuando desea consultar una selección actual que no sea resultado de una consulta previa, pero a partir de un comando como **USE SET**.

Ejemplo

Usted desea consultar los registros que han sido resaltados previamente por el usuario en un formulario de lista. Puedes escribir:

```
USE SET("UserSet")  //reemplazar la selección actual con los registros seleccionados
QUERY SELECTION([Company];[Company]City="New York City";*)
QUERY SELECTION([Company]Type Business="Stock Exchange")
```

Va a encontrar todas las empresas ubicadas en la ciudad de Nueva York, con una actividad de Bolsa, entre la selección inicial del usuario.

QUERY SELECTION BY ATTRIBUTE

QUERY SELECTION BY ATTRIBUTE ({tabla}{;}{conjOp ;} campoObjeto ; rutaAtributo ; opBusq ; valor {; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual devolver una selección de registros o tabla por defecto si se omite
conjOp	Operador	➡	Operador de conjunción a utilizar ara unir múltiples búsquedas (si las hay)
campoObjeto	Campo	➡	Campo objeto para buscar atributos
rutaAtributo	Cadena	➡	Nombre o ruta de atributo
opBusq	Operador, Cadena	➡	Operador de búsqueda (comparador)
valor	Texto, Número, Fecha, Hora	➡	Valor a comparar
*	Operador	➡	Continuar bandera de búsqueda

Descripción

QUERY SELECTION BY ATTRIBUTE trabaja y realiza las mismas acciones que **QUERY BY ATTRIBUTE**. La diferencia entre estos dos comandos es el alcance de la búsqueda:

- **QUERY BY ATTRIBUTE** busca los registros entre todos los registros de la tabla.
- **QUERY SELECTION BY ATTRIBUTE** busca los registros entre los registros seleccionados actualmente en la tabla.

QUERY SELECTION BY ATTRIBUTE busca los registros en *tabla*. El comando **QUERY SELECTION BY ATTRIBUTE** cambia la selección actual de *tabla* para el proceso actual y vuelve el primer registro de la nueva selección el registro actual.

Para más información, consulte la descripción del comando **QUERY BY ATTRIBUTE**.

El comando **QUERY SELECTION BY ATTRIBUTE** es útil cuando una búsqueda no se puede definir mediante una combinación de **QUERY BY ATTRIBUTE** (e incluso **QUERY**) llamadas junto con el parámetro *. Por lo general, este es el caso cuando se desea consultar una selección actual que no sea resultado de una búsqueda previa, pero a partir de un comando como **USE SET**.

Ejemplo

Usted quiere encontrar personas con una edad entre 20 y 30, entre los registros que anteriormente fueron resaltados por el usuario:

```
USE SET("UserSet")  //crea una nueva selección actual
QUERY SELECTION BY ATTRIBUTE([People];[People]OB_Info;"age";>;20;*)
QUERY SELECTION BY ATTRIBUTE([People];&;[People]OB_Info;"age";<;30)  //dispara la búsqueda
```

QUERY SELECTION BY FORMULA

QUERY SELECTION BY FORMULA (tabla {; formula})

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla en la cual efectuar la búsqueda en la selección actual
formula	Booleano	⇒	Fórmula de búsqueda

Descripción

El comando **QUERY SELECTION BY FORMULA** busca registros en tabla. busca registros en tabla aplicando QUERY BY FORMULA a cada registro de la selección **QUERY SELECTION BY FORMULA** modifica la selección actual de tabla para el proceso actual y hace del primer registro el nuevo registro actual.

QUERY SELECTION BY FORMULA funciona de la misma forma que **QUERY BY FORMULA**. La diferencia entres estos dos comandos es el alcance de la búsqueda:

- **QUERY BY FORMULA** busca registros entre todos los registros de la tabla.
- **QUERY SELECTION BY FORMULA** efectúa su búsqueda únicamente en los registros de la selección actual de la tabla.

Para mayor información, consulte la descripción del comando **QUERY BY FORMULA**.

QUERY SELECTION WITH ARRAY

QUERY SELECTION WITH ARRAY (campoObjetivo ; array)

Parámetro	Tipo		Descripción
campoObjetivo	Campo	➡	Campo utilizado para comparar los valores
array	Array	➡	Array de valores buscados

Descripción

El comando **QUERY SELECTION WITH ARRAY** busca en la tabla del campo pasado como primer parámetro los registros para los cuales el valor de campoObjetivo es igual a al menos uno de los valores de los elementos en el *array*. Los registros encontrados constituyen la nueva selección actual.

QUERY SELECTION WITH ARRAY funciona de la misma forma que **QUERY WITH ARRAY**. La diferencia entre estos dos comandos es el alcance de la búsqueda:

- **QUERY WITH ARRAY** busca en todos los registros de la tabla de *campoObjetivo*.
- **QUERY SELECTION WITH ARRAY** busca únicamente en los registros de la selección actual de la tabla *campoObjetivo*.

Para mayor información, consulte la descripción del comando **QUERY WITH ARRAY**.

QUERY WITH ARRAY

QUERY WITH ARRAY (campoObjetivo ; array)

Parámetro	Tipo		Descripción
campoObjetivo	Campo	➡	Campo utilizado para comparar los valores
array	Array	➡	Array de los valores buscados

Descripción

El comando **QUERY WITH ARRAY** busca en la tabla del campo pasado en el primer parámetro todos los registros para los cuales el valor de *campoObjetivo*, es igual al menos a uno de los valores de los elementos en *Array*. Los registros encontrados constituyen la nueva selección actual.

Este comando le permite construir rápida y simplemente una búsqueda en múltiples valores.

Notas:

• Este comando no puede utilizarse con campos de tipo Imagen, subcampo y BLOB.
• *campoObjetivo* y *Array* deben ser del mismo tipo. Excepción: puede utilizar un array de tipo Entero largo con un campo de tipo Hora.

Ejemplo

El siguiente ejemplo le permite recuperar los registros de clientes franceses y americanos:

```
ARRAY STRING(2;ArrayBusqueda;2)
ArrayBusqueda{1}:="FR"
ArrayBusqueda{2}:="US"
QUERY WITH ARRAY([Clientes]Paises;ArrayBusqueda)
```

SET QUERY AND LOCK

SET QUERY AND LOCK (bloq)

Parámetro	Tipo		Descripción
bloq	Booleano	⇒	True = bloquear los registros encontrados por las búsquedas False = No bloquear registros

Descripción

El comando **SET QUERY AND LOCK** permite solicitar el bloqueo automático de los registros encontrados por todas las búsquedas que siguen el llamado de este comando en la transacción actual. Esto significa que los registros no pueden ser modificados por un proceso diferente al proceso actual entre una búsqueda y la manipulación de resultados.

Por defecto, los registros encontrados por las búsquedas no están bloqueados. Pase **True** en el parámetro *bloq* para activar el bloqueo.

Este comando debe imperativamente utilizarse al interior de una transacción. Si se llama fuera de este contexto, se genera un error. Esto permite un mejor control del bloqueo de registros. Los registros encontrados permanecerán bloqueados hasta que la transacción termine (validada o cancelada). Después de que la transacción se completa, todos los registros se desbloquean.

Los registros están bloqueados para todas las tablas en la transacción actual.

Cuando una instrucción SET QUERY AND LOCK(True) ha sido ejecutada, los comandos de búsqueda (por ejemplo QUERY) adoptan un funcionamiento específico si se encuentra un registro bloqueado:

- La búsqueda se detiene y la variable sistema OK toma el valor 0,
- La selección actual queda vacía,
- El conjunto sistema LockedSet contiene el registro bloqueado que causó que la búsqueda se detuviera.

Por lo tanto, en este contexto es necesario probar el conjunto LockedSet definido después de una búsqueda infructuosa (selección actual vacía y/o variable OK en 0) para determinar la causa de la falla.

Llame **SET QUERY AND LOCK** (False) con el fin de desactivar el mecanismo posteriormente.
SET QUERY AND LOCK modifica únicamente el comportamiento de los comandos de búsqueda en otras palabras:

- **QUERY**
- **QUERY SELECTION**
- **QUERY BY EXAMPLE**
- **QUERY BY FORMULA**
- **QUERY BY SQL**
- **QUERY SELECTION BY FORMULA**
- **QUERY SELECTION WITH ARRAY**
- **QUERY WITH ARRAY**
- **QUERY BY ATTRIBUTE**
- **QUERY SELECTION BY ATTRIBUTE**

Sin embargo, **SET QUERY AND LOCK** no afecta los otros comandos que modifican la selección actual tales como **ALL RECORDS**, **RELATE MANY**, etc.

Ejemplo

En este ejemplo, no es posible borrar un cliente que habrías sido pasado de la categoría "C" a la categoría "A" en otro proceso entre **QUERY** y **DELETE SELECTION**:

```
START TRANSACTION
SET QUERY AND LOCK(True)
QUERY([Clientes];[Clientes]Categoria=C)
  `En este momento, los registros encontrados son bloqueados automáticamente para todos los otros procesos
DELETE SELECTION([Clientes])
SET QUERY AND LOCK(False)
VALIDATE TRANSACTION
```

Gestión de errores

Si el comando no se llama en el contexto de una transacción, se genera un error.

SET QUERY DESTINATION

SET QUERY DESTINATION (destinoTipo {; destinoObjeto {; destPunt}})

Parámetro	Tipo		Descripción
destinoTipo	Entero largo	➡	0 = selección actual, 1 = conjunto, 2 = selección temporal, 3 = variable
destinoObjeto	Cadena, Variable	➡	Nombre del conjunto o Nombre de la selección temporal o variable
destPunt	Puntero	➡	Puntero a la variable local si destinoTipo=3

Descripción

El comando **SET QUERY DESTINATION** permite indicar a 4D donde ubicar el resultado de todas las búsquedas posteriores a la llamada a este comando en el proceso actual.

Especifique el tipo de destino en el parámetro *destinoTipo.* 4D ofrece las siguientes constantes predefinidas, que se encuentran en el tema **Destinos de búsqueda**:

Constante	Tipo	Valor
Into current selection	Entero largo	0
Into named selection	Entero largo	2
Into set	Entero largo	1
Into variable	Entero largo	3

Especifique el destino de la búsqueda en el parámetro opcional *destinoObjeto* de acuerdo a la siguiente tabla:

Parámetro destinoTipo	Parámetro destinoObjeto
0 (selección actual)	Omite el parámetro
1 (conjunto)	Pasa el nombre de un conjunto (existente o a crear)
2 (selección temporal)	Pasa el nombre de la selección temporal (existente o a crear)
3 (variable)	Pasa una variable numérica (existente o a crear)

- Con:

```
SET QUERY DESTINATION(Into current selection)
```

Los registros encontrados por la búsqueda se colocarán en la selección actual de la tabla en la cual se efectúa la búsqueda.

- Con:

```
SET QUERY DESTINATION(Into set;"miConjunto")
```

Los registros encontrados por la búsqueda se ubicarán en el conjunto *"miConjunto"*. La selección actual y el registro actual de la tabla en la cual realiza la búsqueda permanecen iguales.

- Con:

```
SET QUERY DESTINATION(Into named selection;"miSeleccionTemporal")
```

Los registros encontrados por la búsqueda se ubicarán en la selección temporal *"miSeleccionTemporal"*. La selección actual y el registro actual para la tabla en la que se efectúa la búsqueda permanecen iguales.

Notas:

- Si la selección temporal no existe de antemano, se creará automáticamente al final de la búsqueda.
- Este comando administra las selecciones temporales como el comando **CUT NAMED SELECTION**: sólo se conservan las referencias. Una vez se utiliza la selección temporal, ya no existe.

Con:

```
SET QUERY DESTINATION(Into variable;$vlResult)
```

O:

```
SET QUERY DESTINATION(Into variable;"";->$vlResult)
```

Nota: este segunda sintaxis facilita el uso conjunto del comando con **GET QUERY DESTINATION**.
El número de registros encontrado por la búsqueda se ubicará en la variable *$vlResult*. La selección actual y el registro actual para la tabla en la que se efectúa la búsqueda permanecen iguales.

Advertencia: SET QUERY DESTINATION afecta todas las búsquedas siguientes en el proceso actual. RECUERDE siempre compensar una llamada a **SET QUERY DESTINATION** (donde *destinoTipe#0*) con una llamada a **SET QUERY DESTINATION**(0) para restaurar el modo de búsqueda estándar.

SET QUERY DESTINATION cambia únicamente el comportamiento de los comandos de búsqueda, es decir:

- **QUERY**
- **QUERY SELECTION**
- **QUERY BY EXAMPLE**
- **QUERY BY FORMULA**
- **QUERY BY SQL**
- **QUERY SELECTION BY FORMULA**
- **QUERY SELECTION WITH ARRAY**
- **QUERY WITH ARRAY**
- **QUERY BY ATTRIBUTE**
- **QUERY SELECTION BY ATTRIBUTE**

Por otra parte, **SET QUERY DESTINATION** no afecta otros comandos que modifican la selección actual de la tabla como **ALL RECORDS**, **RELATE MANY**, etc.

Ejemplo 1

Se crea un formulario que muestra los registros de la tabla *[Libreta telefonica]*. Se crea un objeto de tipo pestaña llamado *asRolodex* (con una pestaña para cada letra del alfabeto) y un subformulario que muestra los registros de la tabla *[Libreta telefonica]*. Al elegir una pestaña, muestra los registros que corresponden a la letra.

En su aplicación, la tabla *[Libreta telefonica]* contiene un conjunto de de datos estáticos, de manera que no necesita realizar una búsqueda cada vez que selecciona una pestaña. De esta manera, puede ahorrar tiempo precioso al ejecutar las búsquedas.

Para hacer esto, puede redireccionar sus búsquedas en las selecciones temporales para reutilizarlas cuando sea necesario. Escriba el método de objeto de la pestaña *asRolodex* como se indica a continuación:

```
   ` Método de objeto asRolodex
Case of
   :(Form event=On Load)
   ` Antes de que el formulario aparezca en la pantalla,
   ` inicializar el rolodex y el array de booleanos que
   ` nos indica si una búsqueda para la letra correspondiente
   ` ha sido realizada o no
      ARRAY STRING(1;asRolodex;26)
      ARRAY BOOLEAN(abQueryDone;26)
      For($vlElem;1;26)
         asRolodex{$vlElem}:=Char(64+$vlElem)
         abQueryDone{$vlElem}:=False
      End for

   :(Form event=On Clicked)
   ` Cuando un usuario hace clic en la pestaña, verificar si la búsqueda correspondiente
   ` ha sido realizada o no
      If(Not(abQueryDone{asRolodex}))
   ` Si no, redireccionar la próxima búsqueda a una selección temporal
         SET QUERY DESTINATION(Into named selection;"temp")
   ` Efectuar la búsqueda
         QUERY([Libreta telefonica];[Libreta telefonica]Last name=asRolodex{asRolodex}+"@")
   ` Restaurar el modo de búsqueda estándar
         SET QUERY DESTINATION(Into current selection)
   ` Utilizar los registros encontrados
         USE NAMED SELECTION("temp")
         COPY NAMED SELECTION([Phone book];"Rolodex+asRolodex{asRolodex})
   ` La próxima vez que seleccionemos esta letra, no realizaremos la búsqueda nuevamente
         abQueryDone{asRolodex}:=True
      Else
   ` Utilice la selección temporal existente para mostrar los registros correspondientes a la letra seleccionada
         USE NAMED SELECTION("Rolodex"+asRolodex{asRolodex}
      End if

   :(Form event=On Unload)
   ` Luego el formulario desaparece de la pantalla
   ` Borrar las selecciones temporales
      For($vlElem;1;26)
         If(abQueryDone{$vlElem})
            CLEAR NAMED SELECTION("Rolodex"+asRolodex{$vlElem})
         End if
      End for
   ` Borrar los dos arrays que ya no necesitamos
      CLEAR VARIABLE(asRolodex)
      CLEAR VARIABLE(abQueryDone)
End case
```

Ejemplo 2

El método de proyecto **ValoresUnicos** en este ejemplo le permite verificar si los valores son únicos para los campos en una tabla. El registro actual puede ser un registro existente o un registro nuevo.

```
` Método de proyecto ValoresUnicos
` ValoresUnicos ( Puntero ; Puntero { ; Puntero... } ) -> Booleano
` ValoresUnicos ( ->Tabla ; ->Campo { ; ->Campo2... } ) -> Yes o No

C_BOOLEAN($0)
C_POINTER($1)
C_LONGINT($vlCampo;$vlNbCampos;$vlEncontrado;$vlRegistroActual)
$vlNbCampos:=Count parameters-1
$vlRegistroActual:=Record number($1->)
If($vlNbCampos>0)
   If($vlRegistroActual#-1)
      If($vlRegistroActual<0)
` El registro actual es un nuevo registro que no ha sido guardado (número de registro -3);
` por lo tanto podemos detener la búsqueda tan pronto como se encuentre al menos un registro
         SET QUERY LIMIT(1)
      Else
` El registro actual es un registro existente;
` por lo tanto podemos detener la búsqueda tan pronto como se encuentren al menos dos registros.
         SET QUERY LIMIT(2)
      End if
` La búsqueda devolverá su resultado en $vlFound
` sin cambiar el registro actual ni la selecciónа actual
      SET QUERY DESTINATION(Into variable;$vlFound)
` Construir la búsqueda de acuerdo al número de campos especificados
      Case of
         :($vlNbCampos=1)
            QUERY($1->;$2->=$2->)
         :($vlNbCampos=2)
            QUERY($1->;$2->=$2->;*)
            QUERY($1->;&;$3->=$3->)
         Else
            QUERY($1->;$2->=$2->;*)
            For($vlCampo;2;$vlNbCampos-1)
               QUERY($1->;&;${1+$vlCampo}->=${1+$vlCampo}->;*)
            End for
            QUERY($1->;&;${1+$vlNbCampos}->=${1+$vlNbCampos}->)
      End case
      SET QUERY DESTINATION(Into current selection)  ` Restaurar el modo de búsqueda estándar
      SET QUERY LIMIT(0)  ` No hay límites de búsquedas
` Procesar el resultado de la búsqueda
      Case of
         :($vlEncontrado=0)
            $0:=True  ` No hay valores duplicados
         :($vlEncontrado=1)
            If($vlRegistroActual<0)
               $0:=False  ` Se encontró un registro existente con los mismos valores que el nuevo registro
            Else
               $0:=True  ` No hay valores duplicado, encontramos el mismo registro
            End if
         :($vlEncontrado=2)
            $0:=False  ` Cualquiera que sea el caso, los valores están duplicados
      End case
   Else
      If(◇DebugOn)  ` No tiene sentido; señálelo durante la versión de desarrollo
         TRACE  ` ¡ATENCIÓN! Este método es llamado sin registro actual
      End if
      $0:=False  ` No es posible garantizar el resultado
   End if
Else
   If(◇DebugOn)  ` No tiene sentido; señálelo si la versión de desarrollo
      TRACE  ` ¡ATENCIÓN! Este método es llamado sin condición de búsqueda
   End if
   $0:=False  ` No puedo garantizar el resultado
End if
```

Después de implementar este método de proyecto en su aplicación, puede escribir:

```
  ` ...
If(ValoresUnicos(->[Contactos];->[Contactos]Empresa;->[Contactos]Apellido;->[Contactos]Nombre))
  ` Realice acciones apropiadas para el registro que tiene valores únicos
Else
   ALERT("Ya existe un contacto con este nombre para esta empresa.")
End if
  ` ...
```

SET QUERY LIMIT

SET QUERY LIMIT (limite)

Parámetro	Tipo		Descripción
limite	Entero largo	➡	Número límite de registros ó 0 para ilimitado

Descripción

SET QUERY LIMIT permite pedirle a 4D detener todas la búsquedas posteriores en el proceso actual tan pronto encuentre el número de registros definido en *limite*.

Si por ejemplo, *limite* es igual a 1, las búsquedas se detendrán tan pronto como un registro coincida con la condiciones de la búsqueda.

Para que las búsquedas sean ilimitadas de nuevo, llame **SET QUERY LIMIT** nuevamente con *limite* igual a 0.

Advertencia: SET QUERY LIMIT afecta todas las búsquedas posteriores en el proceso actual. RECUERDE siempre asociar una llamada a **SET QUERY LIMIT(limite)** (donde *limite>0*) con una llamada a **SET QUERY LIMIT(0)** para restablecer las búsquedas ilimitadas.

SET QUERY LIMIT cambia el comportamiento de los comandos de búsqueda:

- **QUERY**
- **QUERY SELECTION**
- **QUERY BY EXAMPLE**
- **QUERY BY FORMULA**
- **QUERY BY SQL**
- **QUERY SELECTION BY FORMULA**
- **QUERY SELECTION WITH ARRAY**
- **QUERY WITH ARRAY**
- **QUERY BY ATTRIBUTE**
- **QUERY SELECTION BY ATTRIBUTE**

Por otra parte, **SET QUERY LIMIT** no afecta los otros comandos que puedan modificar la selección actual de una tabla como **ALL RECORDS**, **RELATE MANY**, etc.

Ejemplo 1

Para efectuar una búsqueda que corresponda a la fórmula "...encontrar diez clientes cuyas ventas sean superiores a $1 M...", escriba el siguiente código:

```
SET QUERY LIMIT(10)
QUERY([Clientes];[Clientes]Ventas>1000000)
SET QUERY LIMIT(0)
```

Ejemplo 2

Ver el segundo ejemplo del comando **SET QUERY DESTINATION**.

Cadenas de caracteres

- Símbolos de referencia de caracteres
- Etiquetas de transformación 4D
- Change string
- Char
- Character code
- CONVERT FROM TEXT
- Convert to text
- Delete string
- Get localized string
- GET TEXT KEYWORDS
- Insert string
- Length
- Lowercase
- Match regex
- Num
- Position
- Replace string
- Split string
- String
- Substring
- Uppercase
- *_o_Convert case*
- *_o_ISO to Mac*
- *_o_Mac to ISO*
- *_o_Mac to Win*
- *_o_Win to Mac*

Símbolos de referencia de caracteres

Introducción

Los símbolos de índice de cadena son los siguientes: **[[...]]**
Estos símbolos se utilizan para designar un carácter particular en una cadena. Esta sintaxis permite referenciar un carácter en un campo o una variable de tipo Alfa o Texto.
Si los símbolos de referencia de caracteres aparecen a la izquierda del operador de asignación (:=), se asigna un carácter a la posición referenciada en la cadena. Por ejemplo, si *vsNombre* no es una cadena vacía, la siguiente línea pasa el primer carácter de *vsNombre* a mayúsculas:

```
If(vsName#"")
   vsName[[1]]:=Uppercase(vsName[[1]])
End if
```

Si los símbolos de referencia aparecen en una expresión, devuelven el carácter (al cual hacen referencia) como una cadena de un caracter. Por ejemplo:

```
   ` El siguiente ejemplo prueba si el último carácter de vtText es el carácter "@"
If(vtText#"")
   If(Character code(Substring(vtText;Length(vtText);1))=At sign)
   ` ...
   End if
End if

   ` Utilizando la sintaxis de referencia de los caracteres, escriba de una manera sencilla:
If(vtText#"")
   If(Character code(vtText[[Length(vtText)]])=At sign)
   ` ...
   End if
End if
```

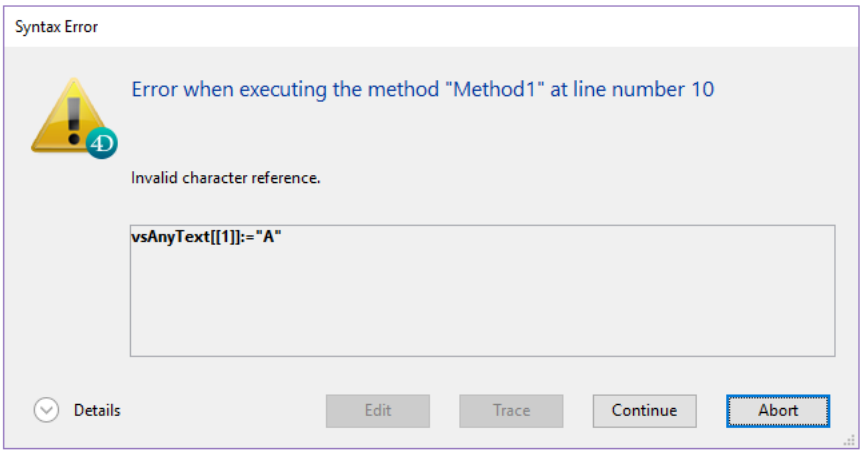
Nota avanzada sobre la referencia de los caracteres inválidos

Cuando utiliza los símbolos de referencia de caracteres, debe direccionar a los caracteres existentes en la cadena de la misma forma que direcciona los elementos de un array. Por ejemplo si direcciona el carácter 20 de una variable de cadena, esta variable DEBE contener por lo menos 20 caracteres.

- No respetar esta condición, en modo interpretado, no provoca un error de sintaxis.
- No respetar esta condición, en modo compilado (sin opciones), puede traer una corrupción de memoria, si, por ejemplo, escribe un carácter más allá del final de una cadena o un texto.
- No respetar esta condición en modo compilado, provoca un error cuando se activa el control de ejecución. Por ejemplo, al ejecutar el siguiente código:

```
   ` ¡No hacer esto!
vsAnyText:=""
vsAnyText[[1]]:="A"
```

provocará el siguiente error:



Ejemplo

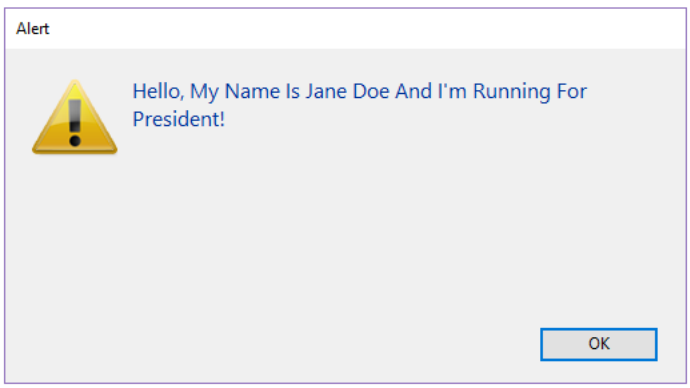
El siguiente método de proyecto coloca en mayúsculas el primer carácter de cada palabra del texto recibido como parámetro y devuelve el texto modificado:

```
` Método de proyecto Pasar a mayúsculas
` Pasar a mayúsculas ( Texto ) -> Text
` Pasar a mayúsculas ( Texto fuente ) -> Texto con letras en mayúsculas

$0:=$1
$vlLen:=Length($0)
If($vlLen>0)
   $0[[1]]:=Uppercase($0[[1]])
   For($vlChar;1;$vlLen-1)
      If(Position($0[[$vlChar]];" !&()-{}:;<>?/,.=+*")>0)
         $0[[$vlChar+1]]:=Uppercase($0[[$vlChar+1]])
      End if
   End for
End if
```

Por ejemplo, la línea:

```
ALERT(Capitalize text("hola, mi nombres es Juan Díaz y soy candidato a presidente!"))
```

Muestra la siguiente alerta:



Etiquetas de transformación 4D

El servidor web 4D le ofrece un conjunto de etiquetas de transformación que permiten insertar las referencias a las expresiones o variables 4D o efectuar diferentes tipos de procesos al interior de un texto fuente, llamado "template". Estas etiquetas son interpretadas durante la ejecución del texto fuente y generan un texto de salida.

Este principio es utilizado por lo general por el servidor web 4D para crear **Páginas semidinamicas**.

Las etiquetas son insertadas generalmente como comentarios tipo HTML (*<!--#Tag Contents-->*). Sin embargo, otros comentarios tales como *<!--Beginning of list-->* también son posibles. Es posible mezclar varios tipos de etiquetas. Por ejemplo, la siguiente estructura HTML es totalmente factible:

```
<HTML> ... <BODY> <!--#4DSCRIPT/PRE_PROCESS-->  (Method call) <!--#4DIF (myvar=1)-->  (If condition)  <!--#4DINCLUDE banner1.html-->  (Subpage insertion) <!--#4DENDIF-->  (End if) <!--#4DIF (mtvar=2)-->  <!--#4DINCLUDE banner2.html--> <!--#4DENDIF--> <!--#4DLOOP [TABLE]-->  (Loop on the current selection) <!--#4DIF ([TABLE]ValNum>10)-->  (If [TABLE]ValNum>10)  <!--#4DINCLUDE subpage.html-->  (Subpage insertion) <!--#4DELSE-->  (Else)  <B>Value: <!--#4DTEXT [TABLE]ValNum--></B><BR>  (Field display) <!--#4DENDIF--> <!--#4DENDLOOP-->  (End for) </BODY> </HTML>
```

Evaluación de los templates

El análisis del contenido de las páginas "templates" se efectúa en dos contextos:

- Utilizando el comando **PROCESS 4D TAGS**; este comando acepta un 'template' como entrada, así como también parámetros (opcionales) y devuelve un texto resultante del procesamiento.
- Utilizando el servidor HTTP integrado de 4D: **Páginas semidinamicas** enviado por medio de los comandos **WEB SEND FILE** (.htm, .html, .shtm, .shtml), **WEB SEND BLOB** (BLOB de tipo text/html), **WEB SEND TEXT** , o llamados utilizando URLs. En este último caso, por razones de optimización, las páginas que tienen sufijos ".htm" and ".html" y NO son analizadas. Para forzar el análisis de las páginas HTML en este caso, debe añadir el sufijo ".shtm" o ".shtml" (por ejemplo, http://www.server.com/dir/page.shtm). Para obtener más información sobre este punto, consulte la sección **Páginas semidinamicas** en el capítulo **Servidor Web**.

Vista general de las etiquetas

La siguiente tabla lista las etiquetas de transformación 4D disponibles. Para más detalles, vea la descripción de las etiquetas a continuación.

Etiqueta	Acción	Ejemplo	Sintaxis $(*)	Comentarios
4DTEXT	Inserta variables y expresiones 4D como texto	<!--#4DTEXT [Customer]Name-->	X	Recomendado si los datos se procesan externamente para evitar inyecciones de código malicioso
4DHTML	Inserta código HTML	<!--#4DHTML -->	X	No se recomienda si los datos se procesan externamente
4DEVAL	Evalúa toda expresión 4D	<!--#4DEVAL a:=20-->	X	No se recomienda si los datos se procesan externamente
4DSCRIPT/	Ejecuta un método 4D con un parámetro	<!--#4DSCRIPT/MyMethod/MyParam-->		
4DINCLUDE	Incluye una página HTML dentro de otra	<!--#4DINCLUDE subpage.html-->		
4DBASE	Designa el archivo utilizado por 4DINCLUDE	<!--#4DBASE ../file/-->		
4DCODE	Inserta código 4D	<!--#4DCODE ALERT(myVar)-->		Soporta CR, LF (bloques de código 4D)
4DIF, 4DELSE, 4DELSEIF, 4DENDIF	Inserta las condiciones en el código dentro de las etiquetas	<!--#4DIF (myVar=1)-->		
4DLOOP, 4DENDLOOP	Inserta bucles en el código dentro de las etiquetas	<!--#4DLOOP [table]-->		Se puede usar con tablas, arrays, métodos, expresiones, punteroArray

(*) Las etiquetas generalmente deben insertarse como comentarios HTML (*<!--#Tag Content-->*) en el texto fuente. Una sintaxis alternativa utilizando $ es posible bajo ciertas condiciones para que las etiquetas devuelvan valores, para que se ajusten a XML. Para más información, consulte **Nueva sintaxis con $ para 4DTEXT, 4DHTML, 4DEVAL**.

Principios de utilización de las etiquetas

Acceso a métodos 4D vía la Web

La ejecución de un método 4D con *4DTEXT*, *4DHTML*, *4DEVAL*, *4DSCRIPT*, *4DIF*, *4DELSEIF* o *4DLOOP* desde una petición web está sujeta al valor del atributo "Available via tags and 4D URLs (4DACTION ...)" "Disponible a través de las etiquetas y URLs 4D (4DACTION ...)" definido en las propiedades del método. Si el atributo no está marcado para el método, no se puede llamar desde una petición web. Para más información sobre este punto, consulte la sección **Seguridad de las conexiones**.

Recursividad del procesamiento

Las etiquetas 4D se interpretan de forma recursiva: 4D siempre intenta reinterpretar el resultado de una transformación y, en caso de que una nueva transformación haya tenido lugar, se realiza una interpretación adicional, y así sucesivamente hasta que el producto obtenido ya no requiera de ninguna transformación adicional. Por ejemplo, dada la siguiente instrucción:

```
<!--#4DHTML [Mail]Letter_type-->
```

Si el campo texto [Mail]Letter_type en sí contiene una etiqueta, por ejemplo *<!--#4DSCRIPT/m_Gender-->*, esta etiqueta se evaluará de forma recursiva después de la interpretación de la etiqueta 4DHTML.

Este principio poderoso cumple la mayoría de las necesidades relacionadas con la transformación de textos. Tenga en cuenta, sin embargo, que en algunos casos esto también puede permitir la inserción de código malicioso. Para más información acerca de este punto, consulte la siguiente sección.

Prevención de inserción de código malicioso

Las etiquetas de transformación 4D aceptan diferentes tipos de datos como parámetros: texto, variables, métodos, nombres de comandos, etc. Cuando estos datos son ofrecidos por su propio código, no hay riesgo de inserción de código malicioso ya que usted controla las entradas. Sin embargo, el código de su base a menudo trabaja con datos que eran, en un momento u otro, introducidos a través de una fuente externa (entrada del usuario, importación, etc.).
En este caso, es aconsejable no utilizar etiquetas de transformación tales como 4DEVAL o 4DSCRIPT, que evalúan los parámetros, directamente con este tipo de datos.
Además, de acuerdo con el principio de la recursividad (ver sección anterior), el código malicioso puede incluir etiquetas de transformación en sí. En este caso, es imprescindible utilizar la etiqueta 4DTEXT.
Imagine, por ejemplo, un campo de formulario web denominado "Name", donde los usuarios deben introducir su nombre. Este nombre se muestra a continuación, utilizando una etiqueta *<!--#4DHTML vName-->* en la página. Si el texto del tipo "*<!--#4DEVAL **QUIT 4D**-->*" se inserta en lugar del nombre, la interpretación de esta etiqueta hará que la aplicación se cierre.

Para evitar este riesgo, sólo puede utilizar la etiqueta 4DTEXT sistemáticamente en este caso. Como esta etiqueta escapa los caracteres HTML especiales, no se reinterpretará cualquier código malicioso recursivo que pueda haber sido insertado. Para hacer referencia al ejemplo anterior, el campo "Nombre" contendrá, en este caso, "*<!--#4DEVAL **QUIT 4D**-->*" que no será transformado.

Identificadores con tokens

Para garantizar la evaluación correcta de las expresiones procesadas a través de etiquetas, independientemente del idioma o la versión en 4D, se recomienda utilizar la sintaxis tokenizada para los elementos cuyo nombre puede variar en versiones (comandos, tablas, campos, constantes). Por ejemplo, para insertar el comando **Current time**, introduzca '**Current time:C178**'. Para más información sobre este punto, consulte la sección **Utilizar tokens en fórmulas**.

Utilización del "." como separador decimal

A partir de v15 R4, 4D utiliza siempre el carácter punto (.) como separador decimal durante la evaluación de una expresión numérica utilizando una etiqueta 4DTEXT, 4DHTML, 4DEVAL o 4DSCRIPT (así como las anteriores etiquetas 4DVAR y 4DHTMLVAR). La configuración regional ahora se ignora en este contexto.

Esta nueva funcionalidad facilitará el mantenimiento y la compatibilidad del código entre las diferentes versiones y lenguajes de 4D.

Por ejemplo, cualquiera que sea la configuración regional:

```
value:=10/4
input:="<!--#4DTEXT value-->"
PROCESS 4D TAGS(input;output)
 // Siempre devuelve 2.5 aunque los ajustes regionales utilizan la ',' como separador
```

Nota de compatibilidad: si su código, convertido desde una versión anterior, evalúa las expresiones numéricas usando las etiquetas 4D con respecto a la configuración regional, es necesario adaptarlo usando el comando **String**:

- Para obtener *valor* con un punto como separador decimal: *<!--#4DTEXT value-->*
- Para obtener el *valor* con un punto decimal definido en la configuración regional: *<!--#4DTEXT String(value)-->*

4DTEXT

Sintaxis: *<!--#4DTEXT VarName-->* o *<!--#4DTEXT 4DExpression-->*
Sintaxis alternativa: *$4DTEXT(VarName)* o *$4DTEXT(4DExpression)* (ver)

La etiqueta *<!--#4DTEXT NombreVar-->* le permite insertar una referencia a una variable o a una expresión 4D que devuelve un valor. Por ejemplo, si escribe (en una página HTML):

<P>Bienvenido a *<!--#4DTEXT vtNomSitio-->*!</P>

El valor de la variable 4D *vtNomSitio* se insertará en la página HTML en el momento de su envío. Este valor se inserta en forma de texto simple, los caracteres HTML especiales tales como ">" se escapan automáticamente.

También puede insertar las expresiones 4D utilizando la etiqueta *4DTEXT*. Y puede por ejemplo insertar directamente el contenido de un campo (*<!--#4DTEXT [nomTabla]nomCampo-->*), un elemento de array (*<!--#4DTEXT tabarr{1}-->*) o un método que devuelve un valor (*<!--#4DTEXT mimetodo-->*). La conversión de la expresión sigue las mismas reglas que la conversión de una variable. Además, la expresión debe cumplir con las reglas de sintaxis de 4D.

En caso de error de evaluación, el texto insertado será de la forma "*<!--#4DTEXT myvar-->* : ## error # error code".

Notas:

- Debe utilizar las variables proceso.
- Es posible mostrar el contenido de un campo imagen. Por el contrario no es posible mostrar el contenido de un elemento de array imagen.
- Es posible mostrar los contenidos de un campo objeto, por intermedio de una fórmula 4D. Por ejemplo, puede escribir *<!--#4DTEXT **OB Get**:C1224([Rect]Desc;\"color\")-->*.
- Por razones de seguridad, se recomienda utilizar esta etiqueta para procesar datos introducidos desde el exterior de la aplicación, para evitar la inserción de código malicioso (ver la sección [#title id="2850" anchor="2626211"/] abajo).
- Por lo general usted trabajará con variables Texto. Sin embargo, puede utilizar variables BLOB. Debe generar el BLOB en modo Texto sin longitud.

4DHTML

Sintaxis: *<!--#4DHTML VarName-->* o *<!--#4DHTML 4DExpression-->*
Sintaxis alternativa: *$4DHTML(VarName)* o *$4DHTML(4DExpression)* (ver)

Como la etiqueta **4DTEXT**, esta etiqueta permite evaluar una variable o una expresión 4D que devuelve un valor, e insertarla como una expresión HTML. A diferencia de la etiqueta **4DTEXT**, esta etiqueta no convierte los caracteres especiales HTML tales como ">").

Por ejemplo, estos son los resultados del procesamiento de la variable texto 4D *mivar* con las etiquetas disponibles:

Valor de mivar	Etiquetas	Resultado
mivar:="<B>"	*<!--#4DTEXT mivar-->*	<B>
mivar:="<B>"	*<!--#4DHTML mivar-->*	<B>

En caso de un error de evaluación, el texto insertado será de la forma "*<!--#4DHTML mivar-->* : ## error # error code".

Nota: por razones de seguridad, se recomienda utilizar la etiqueta **4DTEXT** cuando procese datos introducidos desde fuera de la aplicación para evitar la inserción de código malicioso (ver la sección [#title id="9043" anchor="2626229"/] a continuación).

4DEVAL

Sintaxis: *<!--#4DEVAL VarName--> o <!--#4DEVAL 4DExpression-->*
Sintaxis alternativa: *$4DEVAL(VarName)* or *$4DEVAL(4DExpression*) (ver)

La etiqueta 4DEVAL permite evaluar una variable o una expresión 4D. Al igual que la etiqueta 4DHTML existente, 4DEVAL no escapa los caracteres HTML al regresar del texto. Sin embargo, a diferencia de 4DHTML o 4DTEXT, 4DEVAL le permite ejecutar cualquier instrucción 4D válida, incluyendo asignaciones y expresiones que no devuelven ningún valor.

Por ejemplo, puede ejecutar:

```
$input:="<!--#4DEVAL a:=42-->" //asignación
$input:=$input+"<!--#4DEVAL a+1-->" //cálculo
PROCESS 4D TAGS($input;$output)
 //$output = "43"
```

En caso de error durante la interpretación, el texto insertado será de la forma: "*<!--#4DEVAL expr-->*: ## error # código de error".

Nota: por razones de seguridad, se recomienda utilizar la etiqueta **4DTEXT** cuando se procesan datos introducidos desde fuera de la aplicación, para evitar la inserción de código malicioso (ver la sección [#title id="9043" anchor="2626229"/] a continuación).

4DSCRIPT/

Sintaxis: *<!--#4DSCRIPT/NombreMetodo/Param-->*

La etiqueta *4DSCRIPT* le permite ejecutar métodos 4D en el momento del procesamiento del template. La presencia del comentario HTML *<!--#4DSCRIPT/MiMetodo/MiParam-->* provoca la ejecución del método ***MiMetodo*** con el parámetro *MiParam* como cadena en *$1*.

Nota: si la etiqueta se llama en el contexto de un proceso web, cuando se carga la página, 4D llama al **Método de base On Web Authentication** (si existe). Si devuelve **True**, 4D ejecuta el método.

El método devuelve texto en *$0*. Si la cadena comienza con el carácter de código 1, se considera como HTML (el mismo principio que para la etiqueta *4DHTML*).

El análisis de los contenidos de la página se realiza con **WEB SEND FILE** (.htm, .html, .shtm, .shtml) o se llama **WEB SEND BLOB** (blob de tipo texto/html).
Recuerde que en modo no contextual, el análisis también se hace cuando un URL apunta a un archivo que tiene una extensión ".shtm" o ".shtml" (por ejemplo http://www.server.com/dir/page.shtm).
Por ejemplo, inserte en una página web semi dinámica el comentario "Hoy es*<!--#4DSCRIPT/MIMETH/MIPARAM-->*". Cuando se carga la página, 4D llama al **Método de base On Web Authentication** (si existe), luego llama al método ***MIMET*** y pasa la cadena "/MIPARAM" como parámetro *$1*.

El método devuelve texto en *$0* (por ejemplo "12/31/03"); la expresión "Hoy es *<!--#4DSCRIPT/MIMETH/MIPARAM-->*" se convierte en "Hoy es 12/31/03".

El código del método MIMET es:

```
 //MYMETH
C_TEXT($0;$1) //Estos parámetros debe ser declararse siempre
$0:=String(Current date)
```

Nota: un método llamado por *4DSCRIPT* no debe llamar elementos de interfaz (**DIALOG**, **ALERT**...).

Como 4D ejecuta los métodos en su orden de aparición, es muy posible llamar a un método que define el valor de muchas variables que están referenciadas más adelante en el documento, cualquiera que sea el modo que esté utilizando.

Nota: puede insertar tantos comentarios *<!--#4DSCRIPT...-->* como quiera en un template.

4DINCLUDE

Sintaxis: *<!--#4DINCLUDE Ruta-->*

Esta etiqueta permite incluir, en una página HTML, otra página HTML (designada por el parámetro *ruta*). Por defecto sólo se incluye el cuerpo de la página HTML especificada, es decir el contenido entre las etiquetas <body> y </body> (las etiquetas mismas no se incluyen). Esto le permite evitar conflictos relacionados con las etiquetas presentes en los encabezados. Sin embargo si la página HTML especificada no contiene etiquetas <body></body>, se incluye la página completa. Usted debe verificar la consistencia de las etiquetas.

El comentario *<!--#4DINCLUDE -->* es muy útil en combinación con las pruebas (*<!--#4DIF-->*) o los bucles (*<!--#4DLOOP-->*). Es muy conveniente incluir etiquetas en función de un criterio o de manera aleatoria.
Al momento de la inclusión, sin importar la extensión del nombre del archivo, 4D analiza la página llamada y luego inserta los contenidos (modificados o no) en la página que origina la llamada *4DINCLUDE*.

La ubicación en la caché web de una página incluida con el comentario *<!--#4DINCLUDE -->* responde a las mismas reglas que las páginas llamadas vía un URL o enviadas con el comando **WEB SEND FILE**.

Pase en *ruta*, la ruta de acceso al documento a incluir. **Advertencia**: en el caso de la etiqueta 4DINCLUDE, la ruta de acceso es relativa al documento en curso de análisis, es decir el documento "padre". Utilice el carácter barra oblicua (/) como separador de carpetas y los dos puntos (..) para subir un nivel (sintaxis HTML).

Notas:

- cuando utiliza la etiqueta *4DINCLUDE* con el comando **PROCESS 4D TAGS**, la carpeta por defecto es la carpeta que contiene el archivo de estructura de la base.
- Puede modificar la carpeta por defecto utilizada por la etiqueta *4DINCLUDE* en la página actual, utilizando la etiqueta *<!--#4DBASE -->* (ver a continuación).

El número de *<!--#4DINCLUDE ruta-->* dentro de una página es ilimitado. Sin embargo, las llamadas a *<!--#4DINCLUDE ruta-->* sólo pueden realizarse a un nivel. Esto significa que, por ejemplo, no puede insertar el comentario *<!--#4DINCLUDE midoc3.html-->* en el cuerpo de la página midoc2.html, llamada por *<!--#4DINCLUDE midoc2-->* insertado en midoc1.html.

Además, 4D verifica que las inclusiones no sean recursivas.

En caso de error, el texto insertado es de la forma "*<!--#4DINCLUDE ruta-->* :No se puede abrir el documento".

Ejemplos

```
<<!--#4DINCLUDE subpagina.html--> <!--#4DINCLUDE carpeta/subpagina.html--> <!--#4DINCLUDE ../carpeta/subpagina.html-->
```

4DBASE

Sintaxis: *<!--#4DBASE folderPath-->*

La etiqueta *<!--#4DBASE -->* designa un directorio de trabajo que será utilizado por la etiqueta *<!--#4DINCLUDE-->*.
Cuando se llama en una página web, la etiqueta *<!--#4DBASE -->* modifica todas las llamadas *<!--#4DINCLUDE-->* posteriores en esta página, hasta la próxima *<!--#4DBASE -->*, si la hay. Si la carpeta *<!--#4DBASE -->* se modifica desde un archivo incluido, recupera su valor original desde el archivo padre.

El parámetro *rutaCarpeta* debe contener una ruta de acceso relativa a la página actual y debe terminar con una barra oblicua (/). La carpeta designada debe ubicarse al interior de la carpeta web.
Pase la palabra clave *WEBFOLDER* para restablecer la ruta por defecto (relativa a la página).

Por lo tanto el siguiente código, debe especificar una ruta relativa a cada llamada:

```
<!--#4DINCLUDE subpage.html--> <!--#4DINCLUDE folder/subpage1.html--> <!--#4DINCLUDE folder/subpage2.html--> <!--#4DINCLUDE folder/subpage3.html--> <!--#4DINCLUDE ../folder/subpage.html-->
```

... puede escribirse utilizando la etiqueta *<!--#4DBASE -->*:

```
<!--#4DINCLUDE subpage.html--> <!--#4DBASE folder/--> <!--#4DINCLUDE subpage1.html--> <!--#4DINCLUDE subpage2.html--> <!--#4DINCLUDE subpage3.html--> <!--#4DBASE ../folder/--> <!--#4DINCLUDE subpage.html--> <!--#4DBASE WEBFOLDER-->
```

Ejemplo

Definición de un directorio para la página de inicio utilizando la etiqueta *<!--#4DBASE -->*:

```
/* Index.html */ <!--#4DIF LangFR=True--> <!--#4DBASE FR/--> <!--#4DELSE--> <!--#4DBASE US/--> <!--#4DENDIF--> <!--#4DINCLUDE head.html--> <!--#4DINCLUDE body.html--> <!--#4DINCLUDE footer.html-->
```

En el archivo head.html, la carpeta actual se modifica vía *<!--#4DBASE -->*, sin que cambie su valor en Index.html:

```
/* Head.htm */ /* el directorio de trabajo aquí es relativo al archivo incluido (FR/ o US/) */ <!--#4DBASE Styles/--> <!--#4DINCLUDE main.css--> <!--#4DINCLUDE product.css--> <!--#4DBASE Scripts/--> <!--#4DINCLUDE main.js--> <!--#4DINCLUDE product.js-->
```

4DCODE

La etiqueta 4DCODE le permite insertar un bloque de código multilínea 4D en una plantilla.

Cuando se detecta que una secuencia "<!--#4DCODE" está seguida por un espacio, un carácter CR o un LF, 4D interpreta todas las líneas de código hasta la próxima secuencia "-->". El bloque de código en sí mismo puede contener retornos de carro, saltos de línea, o ambos; será interpretado secuencialmente por 4D.

Por ejemplo, utilizando la etiqueta 4DCODE, puede escribir en una plantilla:

```
<!--#4DCODE
//Inicialización de los parámetros
C_OBJECT:C1216($graphParameters)
OB SET:C1220($graphParameters;"graphType";1)
$graphType:=1
//...votre code ici
If(OB Is defined:C1231($graphParameters;"graphType")) //lenguaje US únicammente
    $graphType:=OB GET:C1224($graphParameters;"graphType")
    If($graphType=7)
        $nbSeries:=1
        If($nbValues>8)
            DELETE FROM ARRAY:C228 ($yValuesArrPtr{1}->;9;100000)
            $nbValues:=8
        End if
    End if
End if
-->
```

Nota: en una etiqueta 4DCODE, el código 4D siempre debe ser escrito utilizando el lenguaje Inglés - Estados Unidos. Por lo tanto, 4DCODE ignora las preferencias de usuario "Utilizar configuración del sistema regional" para el lenguaje 4D (ver **Lenguaje para los comandos y constantes**).

Estas son la funcionalidades de la etiqueta 4DCODE:

- El comando **TRACE** es soportado y activa el depurador 4D, lo que le permite depurar el código de su plantilla.
- Cualquier error se mostrará el diálogo de error estándar que permite al usuario detener la ejecución del código o introducir el modo de depuración.
- El texto entre <!--#4DCODE y --> se divide en líneas que aceptan cualquier convención de final de línea (cr, lf, o crlf).
- El texto se tokeniza dentro del contexto de la base que llamó **PROCESS 4D TAGS**. Esto es importante para el reconocimiento de los métodos de proyecto, por ejemplo.
  Nota: la propiedad del método "Disponible por etiquetas 4D tags y URLs 4DACTION" no se tiene en cuenta (ver también la **Nota de seguridad** más abajo).
- Incluso si el texto siempre utiliza Inglés - Estados Unidos, se recomienda utilizar la sintaxis de token (:Cxxx) para nombres de comandos y de constantes para proteger contra potenciales problemas por comandos o constantes renombrados de una versión de 4D a otra.
  Nota: para más información sobre*:* sintaxis *Cxxx*, consulte la sección **Utilizar tokens en fórmulas**.

Nota de seguridad: el hecho de que las etiquetas 4DCODE puedan llamar a cualquiera de los comandos del lenguaje 4D o a los métodos de proyecto podría ser visto como un problema de seguridad, especialmente cuando la base de datos está disponible a través de HTTP. Sin embargo, ya que ejecuta el código del lado del servidor llamado desde sus propios archivos de plantilla, la etiqueta en sí no representa un problema de seguridad. En este contexto, como para cualquier servidor Web, la seguridad se maneja principalmente a nivel de accesos remotos a los archivos del servidor.

4DIF, 4DELSE, 4DELSEIF y 4DENDIF

Sintaxis: *<!--#4DIF expresion--> {<!--#4DELSEIF expression2-->...<!--#4DELSEIF expresionN-->} {<!--#4DELSE-->} <!--#4DENDIF-->*

Utilizado con los comentarios *<!--#4DELSEIF-->* (opcional), *<!--#4DELSE-->* (opcional) *y <!--#4DENDIF-->*, el comentario *<!--#4DIF expresion-->* ofrece la posibilidad de ejecutar partes de código de manera condicional.

El parámetro *expresion* puede contener toda expresión 4D válida devolviendo un valor booleano. Debe estar entre paréntesis y cumplir con las reglas de sintaxis de 4D.

Los bloques *<!--#4DIF expression-->* ... *<!--#4DENDIF-->* pueden estar anidados en varios niveles. Como en 4D, cada *<!--#4DIF expresion-->* debe tener un *<!--#4DENDIF-->* correspondiente.

En caso de un error de interpretación, el texto "*<!--#4DIF expresion-->*: se esperaba una expresión booleana" se inserta en lugar del contenido situado entre *<!--#4DIF -->* y *<!--#4DENDIF-->*.

De la misma forma, si no hay tantos *<!--#4DENDIF-->* como *<!--#4DIF -->*, el texto "*<!--#4DIF expresion-->*: 4DENDIF esperado" se inserta en lugar de los contenidos ubicados entre *<!--#4DIF -->* y *<!--#4DENDIF-->*.

Utilizando la etiqueta *<!--#4DELSEIF-->*, puede probar un número ilimitado de condiciones. Sólo se ejecuta el contenido que sigue la primera condición evaluada como **True**. Si ninguna condición es verdadera, no se ejecuta ninguna instrucción (si no hay *<!--#4DELSE-->* final).
Puede utilizar una etiqueta *<!--#4DELSE-->* después de la última *<!--#4DELSEIF-->*. Si todas las condiciones son falsas, las instrucciones siguientes a *<!--#4DELSE-->* se ejecutan.

Los siguientes dos códigos son equivalentes.

- Código utilizando únicamente 4DELSE:

```
<!--#4DIF Condition1--> /* Condition1 es true*/ <!--#4DELSE--> <!--#4DIF Condition2--> /* Condition2 es true*/ <!--#4DELSE--> <!--#4DIF Condition3--> /* Condition3 es true */ <!--#4DELSE--> /*Ninguna de las condiciones es verdadera*/ <!--#4DENDIF--> <!--#4DENDIF--> <!--#4DENDIF-->
```

- Código similar utilizando la etiqueta 4DELSEIF:

<!--#4DIF Condition1--> /* Condition1 es true*/ <!--#4DELSEIF Condition2--> /* Condition2 es true*/ <!--#4DELSEIF Condition3--> /* Condition3 es true */ <!--#4DELSE--> /* Ninguna de las condiciones es verdadera*/ <!--#4DENDIF-->

Ejemplo 1

Este ejemplo de código insertado en una página HTML estática muestra una etiqueta diferente de acuerdo al resultado de la expresión vnom#"":

<BODY> ... <!--#4DIF (vnom#"")--> Nombres que comienzan por <!--#4DTEXT vnom-->. <!--#4DELSE--> No se ha encontrado ningún nombre. <!--#4DENDIF--> ... </BODY>

Ejemplo 2

Este ejemplo inserta páginas diferentes en función del usuario conectado:

<!--#4DIF LoggedIn=False--> <!--#4DINCLUDE Login.htm --> <!--#4DELSEIF User="Admin" --> <!--#4DINCLUDE AdminPanel.htm --> <!--#4DELSEIF User="Manager" --> <!--#4DINCLUDE SalesDashboard.htm --> <!--#4DELSE--> <!--#4DINCLUDE ItemList.htm --> <!--#4DENDIF-->

4DLOOP y 4DENDLOOP

Sintaxis: *<!--#4DLOOP condicion--> <!--#4DENDLOOP-->*

Este comentario permite la repetición de una parte de código siempre y cuando se cumpla la condición. La porción está delimitada por *<!--#4DLOOP-->* y *<!--#4DENDLOOP-->*.

Los bloques *<!--#4DLOOP condicion-->* ... *<!--#4DENDLOOP-->* pueden estar imbricados. Como en 4D, cada *<!--#4DLOOP condicion-->* debe tener un *<!--#4DENDLOOP-->* correspondiente.

Hay cinco tipos de condiciones:

- <!--#4DLOOP [tabla]-->

  Esta sintaxis realiza un bucle para cada registro desde la selección actual de la *tabla* en el proceso en actual. La porción de código ubicada entre los dos comentarios se repite para cada registro de la selección actual.

Nota: cuando la etiqueta 4DLOOP se utiliza con una tabla, los registros se cargan en modo sólo lectura.

El ejemplo de código siguiente:

<!--#4DLOOP [Personas]--> <!--#4DTEXT [Personas]Nombre--> <!--#4DVAR [Personas]Apellido-->
 <!--#4DENDLOOP-->

... puede traducirse en lenguaje 4D como:

```
FIRST RECORD([Personas])
While(Not(End selection([Personas])))
   ...
   NEXT RECORD([Personas])
End while
```

- <!--#4DLOOP array-->

Esta sintaxis efectúa in bucle para cada elemento del array. El índice actual del array se incrementa cuando se repite la porción de código.

Nota: esta sintaxis no puede utilizarse con arrays de dos dimensiones. En este caso, es mejor combinar un método con bucles anidados.

El siguiente ejemplo de código siguiente:

<!--#4DLOOP arr_nombres--> <!--#4DTEXT arr_nombres{arr_nombres}-->
 <!--#4DENDLOOP-->

... puede traducirse en lenguaje 4D como:

```
For($Elem;1;Size of array(arr_nombres))
   arr_nombres:=$Elem
   ...
End for
```

- <!--#4DLOOP metodo-->

Esta sintaxis efectúa un bucle mientras el método devuelva True. El método admite un parámetro de tipo Entero largo. Primero se llama con el valor 0 para permitir una fase de inicialización (si es necesaria); luego se llama con los valores 1, 2, 3, ... mientras devuelva True.

Por razones de seguridad, en el contexto de un proceso web **Método base On Web Authentication** puede llamarse, sólo una sola vez, antes de la fase de inicialización (ejecución del método con 0 como parámetro). Si se confirma la autenticación, se lleva a cabo la fase de inicialización.

Advertencia: **C_BOOLEAN**($0) y **C_LONGINT**($1) DEBEN declararse dentro del método por motivos de compilación.

El ejemplo de código siguiente:

```
<!--#4DLOOP mi_metodo--> <!--#4DTEXT var--> <BR> <!--#4DENDLOOP-->
```

... puede traducirse en código 4D como:

```
If(AuthenticationWebOK)
   If(mi_metodo(0))
      $contador:=1
      While(mi_metodo($contador))
         ...
         $contador:=$contador+1
      End while
   End if
End if
```

El método ***mi_metodo*** puede ser de esta forma:

```
C_LONGINT($1)
C_BOOLEAN($0)
If($1=0)
   `Inicialización
   $0:=True
Else
   If($1<50)
      ...
      var:=...
      $0:=True
   Else
      $0:=False  `Detiene el bucle
   End if
End if
```

- **<!--#4DLOOP 4DExpression-->**
  Con esta sintaxis, la etiqueta 4DLOOP efectúa un bucle siempre que la expresión 4D devuelva **True**. La expresión puede ser toda expresión booleana válida y debe contener una parte variable a evaluar en cada bucle para evitar bucles infinitos.
  Por ejemplo, el siguiente código:

```
<!--#4DEVAL $i:=0--> <!--#4DLOOP ($i<4)--> <!--#4DEVAL $i--> <!--#4DEVAL $i:=$i+1--> <!--#4DENDLOOP-->
```

  produce el siguiente resultado:

```
0
1
2
3
```

- **<!--#4DLOOP pointerArray-->**
  En este caso, la etiqueta 4DLOOP funciona como lo hace con un array: hace un bucle para cada elemento del array referenciado por el puntero. El elemento del array actual se incrementa cada vez que la porción de código se repite.
  Esta sintaxis es útil cuando se pasa un puntero array como parámetro al comando **PROCESS 4D TAGS**.
  Ejemplo:

```
ARRAY TEXT($array;2)
$array{1}:="hello"
$array{2}:="world"
$input:="<!--#4DEVAL $1-->"
$input:=$input+"<!--#4DLOOP $2-->"
$input:=$input+"<!--#4DEVAL $2->{$2->}--> "
$input:=$input+"<!--#4DENDLOOP-->"
```

```
PROCESS 4D TAGS($input;$output;"elements = ";->$array)
 // $output = "elements = hello world "
```

En caso de error de evaluación, el texto "*<!--#4DLOOP expression-->*: descripción" se inserta en lugar del contenido ubicado entre *<!--#4DLOOP -->* y *<!--#4DENDLOOP-->*.

Los siguientes mensajes se pueden mostrar:

- Una expresión de tipo no esperado (error estándar);
- Nombre de tabla invalido (error en el nombre de la tabla);
- Se esperaba un array (la variable no es un array o es un array de dos dimensiones);
- El método no existe;
- Error de sintaxis (durante la ejecución del método);
- Error de acceso (usted no tienen los privilegios de acceso apropiados para acceder a la tabla o al método).
- 4DENDLOOP esperado (el número de *<!--#4DENDLOOP-->* no corresponde al de *<!--#4DLOOP -->*).

Nueva sintaxis con $ para 4DTEXT, 4DHTML, 4DEVAL

Varias etiquetas de transformación 4D existentes ahora pueden expresarse utilizando una nueva sintaxis utilizando el signo dólar $:
$4dtag (*expression*) se puede utilizar en lugar de **<!--#4dtag *expression*-->**

Condiciones requeridas
La sintaxis $ sólo se utiliza con ciertas etiquetas y en ciertos contextos de evaluación:

- **Etiquetas**
  Sólo las etiquetas utilizadas para devolver los valores procesados aceptan esta sintaxis:
  - 4DTEXT
  - 4DHTML
  - 4DEVAL
- **Evaluación**
  La sintaxis $ dispara la evaluación de etiquetas por 4D sólo en casos donde la evaluación es explícita:
  - comando **PROCESS 4D TAGS**
  - páginas estáticas **.shtml** servidas directamente por el servidor web por medio de URLs
  - páginas insertadas por medio de la etiqueta 4DINCLUDE

Para las otras etiquetas (4DIF, 4DSCRIPT, etc.) y en otros contextos de evaluación (comandos **WEB SEND FILE**, **WEB SEND BLOB** y **WEB SEND TEXT**), la sintaxis $ no es soportada y es necesario utilizar la sintaxis regular).

Uso

Por ejemplo, puede escribir:

```
$4DEVAL(UserName)
```

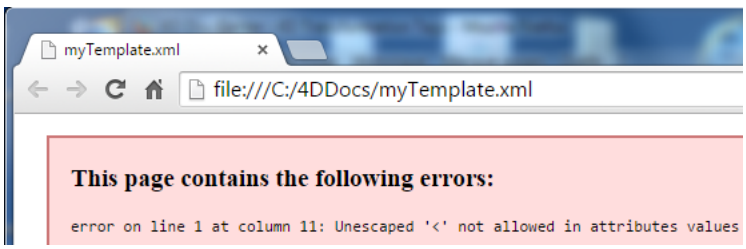
en lugar de:

```
<!--#4DEVAL(UserName)-->
```

La principal ventaja de esta nueva sintaxis es que le permite **escribir plantillas conformes a la norma XML**. Algunos desarrolladores 4D necesitan crear y validar las plantillas basadas en XML usando herramientas de XML estándar. Dado que el carácter "<" no es válido en un valor de atributo XML, no fue posible usar la sintaxis "<!-- -->" para las etiquetas 4D sin romper la sintaxis documento. Por otra parte, evadir el carácter "<" impedirá que 4D interprete las las etiquetas correctamente.

Por ejemplo, el siguiente código podría causar un error de análisis XML debido al primer carácter "<" en el valor del atributo:

```
<line x1="<!--#4DEVAL $x-->" y1="<!--#4DEVAL $graphY1-->"/>
```



Utilizando la sintaxis $, el siguiente código es validado por el analizador:

```
<line x1="$4DEVAL($x)" y1="$4DEVAL($graphY1)"/>
```

Note *que $4dtag* y *<--#4dtag -->* no son estrictamente equivalentes: a diferencia de *<--#4dtag -->*, el procesamiento de *$4dtag* no interpreta etiquetas 4D recursivamente. Las etiquetas $ siempre se evalúan una sola vez y el resultado es considerado como texto plano.

Nota: para más información sobre el procesamiento recursivo, por favor consulte el párrafo .

La razón de esta diferencia es evitar que la inyección de código malicioso. Como se explica en el manual de referencia *Lenguaje*, se recomienda especialmente el uso de las etiquetas 4DTEXT en lugar de las etiquetas 4DHTML al manipular texto de usuario para proteger contra la reinterpretación no deseada de las etiquetas: con 4DTEXT, los caracteres especiales como "<" se "escapan", por lo tanto, ninguna etiqueta 4D el uso de la etiqueta *<!--#4dtag expression -->* sintaxis perderá su significado particular. Sin embargo, como 4DTEXT no "escapa" el símbolo $, decidimos romper el soporte a la recursividad con el fin de evitar la inyección de código malicioso usando la sintaxis *$4dtag (expression)*.

Los siguientes ejemplos muestran el resultado del procesamiento en función de la sintaxis y la etiqueta utilizada:

```
 // ejemplo 1
myName:="<!--#4DHTML QUIT 4D-->"  //inyección maliciosa
input:="My name is: <!--#4DHTML myName-->"
PROCESS 4D TAGS(input;output)
 //4D se cerrará
```

```
 // ejemplo 2
myName:="<!--#4DHTML QUIT 4D-->"  //inyección maliciosa
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
 //la salida es "My name is: &lt;!--#4DHTML QUIT 4D--&gt;"
```

```
 // ejemplo 3
myName:="$4DEVAL(QUIT 4D)"  //inyección maliciosa
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
 //la salida es "My name is: $4DEVAL(QUIT 4D)"
```

Tenga en cuenta que la sintaxis *$4dtag* soporta los pares correspondientes de comillas o paréntesis incluidos. Por ejemplo, supongamos que necesita para evaluar la siguiente cadena compleja (ejemplo teórico):

```
String(1) + "\"(hello)\""
```

Puede escribir:

```
input:="$4DEVAL( String(1)+\"\\\"(hello)\\\"\")"
PROCESS 4D TAGS(input;output)
-->la salida es 1"(hello)"
```

Change string

Change string (fuente ; nuevo ; posicion) -> Resultado

Parámetro	Tipo		Descripción
fuente	Cadena	➡	Cadena original
nuevo	Cadena	➡	Nuevos caracteres
posicion	Entero largo	➡	Posición donde comenzar los cambios
Resultado	Cadena	↩	Cadena resultante

Descripción

Change string devuelve una cadena resultante de cambiar los caracteres, en la cadena *fuente*, a partir de *posicion* con los caracteres en *nuevo*.

Si *nuevo* es una cadena vacía (""), **Change string** devuelve *fuente* sin cambios. **Change string** siempre devuelve una cadena de la misma longitud que *fuente*. Si *posicion* es inferior o superior a la longitud de *fuente*, **Change string** devuelve *fuente*.

Change string se diferencia de **Insert string** en que reemplaza los caracteres en lugar de insertarlos.

Ejemplo

El siguiente ejemplo ilustra el uso de **Change string**. Los resultados se asignan a la variable *vtResult*.

```
vtResult:=Change string("Acme";"CME";2) ` vtResult es igual a "ACME"
vtResult:=Change string("noviembre";"dic";1) ` vtResult es igual a "diciembre"
```

Char

Char (codigoCaracter) -> Resultado

Parámetro	Tipo		Descripción
codigoCaracter	Entero largo	→	Código del caracter
Resultado	Cadena	↩	Caracter representado por codigoCaracter

Descripción

El comando **Char** devuelve el carácter cuyo código es *codigoCaracter*.

Pase un valor UTF-16 (entre 1 y 65535) en *codigoCaracter.*

Tip: el comando **Char** generalmente se utiliza para insertar en el editor de métodos los caracteres que no pueden ser introducidos desde el teclado o que deben ser interpretados como un comando de edición en el editor de métodos.

Ejemplo

El siguiente ejemplo utiliza **Char** para insertar un retorno de carro en el texto de un mensaje de alerta:

```
ALERT("Empleados: "+String(Records in table([Empleados]))+Char(Carriage return)+"Presione Aceptar para continuar.")
```

Character code

Character code (unCaracter) -> Resultado

Parámetro	Tipo		Descripción
unCaracter	Cadena	➡	Caracter para el cual obtener el código
Resultado	Entero largo	⮌	Código del caracter

Descripción

El comando **Character code** devuelve el código Unicode UTF-16 (incluído entre 1 y 65535) de *unCaracter*.

Si hay más de un carácter en la cadena, **Character code** devuelve únicamente el código del primer carácter.
La función **Char** es la contraparte de **Character code**. Devuelve el carácter designado por un código UTF-16.

Ejemplo 1

Los caracteres en mayúsculas y minúsculas se consideran iguales en una comparación. Puede utilizar **Character code** para diferenciar entre los caracteres en mayúsculas y en minúsculas. Por lo tanto, esta línea devuelve True:

```
("A"="a")
```

Por otra parte, esta línea devuelve False:

(Character code("A")=Character code("a"))

Ejemplo 2

Este ejemplo devuelve el código del primer carácter de la cadena *"ABC"*:

```
RecupCod:=Character code("ABC")  ` RecupCod toma el valor 65, el código del carácter de A
```

Ejemplo 3

El siguiente ejemplo prueba los retornos de carro y los tabuladores:

```
For($vlCar;1;Length(vtText))
   Case of
      :(vtText[[$vlCar]]=Char(Carriage return))
   ` Hacer algo
      :(vtText[[$vlCar]]=Char(Tab))
   ` Hacer otra cosa
      :(...)
   ` ...
   End case
End for
```

Cuando se ejecuta muchas veces en textos largos, se ejecutará más rápido, una vez compilado, si se escribe de esta forma:

```
For($vlCar;1;Length(vtText))
   $vlCode:=Character code(vtText[[$vlCar]])
   Case of
      :($vlCode=Carriage return)
   ` Hacer algo
      :($vlCode=Tab)
   ` Hacer otra cosa
      :(...)
   ` ...
   End case
End for
```

El segundo código se ejecuta más rápido por dos razones: sólo referencia un carácter por iteración y utiliza comparaciones de enteros largos en lugar de comparaciones de cadenas para probar los retornos de carro y las tabulaciones. Utilice esta técnica cuando trabaje con códigos comunes tales como CR y TAB.

CONVERT FROM TEXT

CONVERT FROM TEXT (texto4D ; juegoCaracteres ; blobConvertido)

Parámetro	Tipo		Descripción
texto4D	Cadena	➡	Texto expresado en el juego de caracteres actual de 4D
juegoCaracteres	Cadena, Entero largo	➡	Nombre o número del juego de caracteres
blobConvertido	BLOB	⬅	BLOB que contiene el texto convertido

Descripción

El comando **CONVERT FROM TEXT** permite convertir un texto expresado en el juego de caracteres actual de 4D en un texto expresado en otro juego de caracteres.

En el parámetro *texto4D*, pase el texto a convertir. Este texto está expresado en el juego de caracteres de 4D. En la versión 11, 4D utiliza el juego de caracteres Unicode por defecto.

En *juegoCaracteres*, pase el juego de caracteres a utilizar para la conversión. Puede pasar una cadena que contenga el nombre estándar del juego (por ejemplo "ISO-8859-1" o "UTF-8"), o su identificador MIBEnum.

La siguiente es una lista de conjuntos de caracteres soportada por los comandos **CONVERT FROM TEXT** y **Convert to text** :

MIBEnum	**Nombre**(s)
1017	UTF-32
1018	UTF-32BE
1019	UTF-32LE
1015	UTF-16
1013	UTF-16BE
1014	UTF-16LE
106	UTF-8
1012	UTF-7
3	US-ASCII
3	ANSI_X3.4-1968
3	ANSI_X3.4-1986
3	ASCII
3	cp367
3	csASCII
3	IBM367
3	iso-ir-6
3	ISO_646.irv:1991
3	ISO646-US
3	us
2011	IBM437
2011	cp437
2011	437
2011	csPC8CodePage437
2028	ebcdic-cp-us
2028	cp037
2028	csIBM037
2028	ebcdic-cp-ca
2028	ebcdic-cp-n
2028	ebcdic-cp-wt
2028	IBM037
2027	MacRoman
2027	x-mac-roman
2027	mac
2027	macintosh
2027	csMacintosh
2252	windows-1252
1250	MacCE
1250	x-mac-ce
2250	windows-1250
1251	x-mac-cyrillic
2251	windows-1251
1253	x-mac-greek
2253	windows-1253
1254	x-mac-turkish
2254	windows-1254
1256	x-mac-arabic
2256	windows-1256
1255	x-mac-hebrew
2255	windows-1255
1257	x-mac-ce
2257	windows-1257
17	Shift_JIS
17	csShiftJIS
17	MS_Kanji
17	Shift-JIS
39	ISO-2022-JP
39	csISO2022JP
2026	Big5
2026	csBig5
38	EUC-KR
38	csEUCKR
2084	KOI8-R
2084	csKOI8R
4	ISO-8859-1
4	CP819
4	csISOLatin1
4	IBM819

4	iso-ir-100
4	ISO_8859-1
4	ISO_8859-1:1987
4	l1
4	latin1
5	ISO-8859-2
5	csISOLatin2
5	iso-ir-101
5	ISO_8859-2
5	ISO_8859-2:1987
5	l2
5	latin2
6	ISO-8859-3
6	csISOLatin3
6	ISO-8859-3:1988
6	iso-ir-109
6	ISO_8859-3
6	l3
6	latin3
7	ISO-8859-4
7	csISOLatin4
7	ISO-8859-4:1988
7	iso-ir-110
7	ISO_8859-4
7	l4
7	latin4
8	ISO-8859-5
8	csISOLatinCyrillic
8	cyrillic
8	ISO-8859-5:1988
8	iso-ir-144
8	ISO_8859-5
9	ISO-8859-6
9	arabic
9	ASMO-708
9	csISOLatinArabic
9	ECMA-114
9	ISO-8859-6:1987
9	iso-ir-127
9	ISO_8859-6
10	ISO-8859-7
10	csISOLatinGreek
10	ECMA-118
10	ELOT_928
10	greek
10	greek8
10	iso-ir-126
10	ISO_8859-7
10	ISO_8859-7:1987
11	ISO-8859-8
11	csISOLatinHebrew
11	hebrew
11	iso-ir-138
11	ISO_8859-8
11	ISO_8859-8:1988
12	ISO-8859-9
12	csISOLatin5
12	iso-ir-148
12	ISO_8859-9
12	ISO_8859-9:1989
12	l5
12	latin5
13	ISO-8859-10
13	csISOLatin6
13	iso-ir-157
13	ISO_8859-10
13	ISO_8859-10:1992
13	l6

13	latin6
109	ISO-8859-13
111	ISO-8859-15
111	Latin-9
113	GBK
2025	GB2312
2025	csGB2312
2025	x-mac-chinesesimp
2025	x-mac-chinesesimp
2024	Windows-31J
57	GB_2312-80
57	csISO58GB231280

Nota: varias líneas tienen el mismo identificador MIBEnum porque un conjunto de caracteres puede tener más de un nombre (alias).

Para mayor información sobre los nombres de los juegos de caracteres, por favor consulte la siguiente dirección: *http://www.iana.org/assignments/character-sets*

Después de la ejecución del comando, el texto convertido será devuelto en el BLOB *blobConvertido*. Este BLOB podrá ser leído por el comando **Convert to text**.

Variables y conjuntos del sistema

Si el comando ha sido ejecutado correctamente, la variable OK toma el valor 1. De lo contrario, toma el valor 0.

Convert to text

Convert to text (BLOB ; juegoCaracteres) -> Resultado

Parámetro	Tipo		Descripción
BLOB	BLOB	→	BLOB que contiene un texto expresado en un juego de caracteres específico
juegoCaracteres	Cadena, Entero largo	→	Nombre o número de juego de caracteres de blob
Resultado	Texto	←	Contenido del BLOB expresado en el juego de caracteres 4D

Descripción

El comando **Convert to text** convierte el texto contenido en el parámetro *blob* y lo devuelve en texto expresado en el juego de caracteres de 4D. 4D utiliza por defecto el conjunto de caracteres UTF-16.

En *juegoCaracteres*, pase el conjunto de caracteres del texto contenido en el *blob*, el cual será utilizado para la conversión. Si el BLOB contiene texto copiado desde 4D, el texto del BLOB será probablemente UTF-16. Puede pasar una cadena con el nombre estándar del conjunto de caracteres, o una con sus alias (por ejemplo, "ISO-8859-1" o "UTF-8"), o su identificador (entero largo). Para mayor información, consulte la descripción del comando **CONVERT FROM TEXT**.
Convert to text soporta BOMs (Byte Order Marks). Si el conjunto de caracteres especificado es de tipo Unicode (UTF-8, UTF-16 o UTF-32), 4D intenta identificar un BOM entre los primeros bytes recibidos. Si lo detecta, se filtra del resultado y 4D utiliza el conjunto de caracteres definido en lugar del conjunto de caracteres especificado.

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable OK toma el valor 1. De lo contrario, toma el valor 0.

Delete string

Delete string (fuente ; posicion ; numCaracteres) -> Resultado

Parámetro	Tipo		Descripción
fuente	Cadena	➡	Cadena de la cual borrar caracteres
posicion	Entero largo	➡	Primer caracter a borrar
numCaracteres	Entero largo	➡	Número de caracteres a borrar
Resultado	Cadena	⮌	Cadena resultante

Descripción

Delete string borra *numCaracteres* de *fuente*, a partir de *posicion*, y devuelve la cadena resultante.
Delete string devuelve la misma cadena que *fuente* cuando:

- *fuente* es una cadena vacía
- *posicion* es superior a la longitud de *fuente*
- *numCaracteres* es igual a cero (0)

Si *posicion* es inferior a uno, los caracteres son borrados a partir del inicio de la cadena.
Si *posicion* más *numCaracteres* es igual o mayor a la longitud de *fuente*, los caracteres se borran a partir de *posicion* hasta el final de *fuente*.

Ejemplo

El siguiente ejemplo ilustra el uso de **Delete string**. Los resultados se asignan a la variable *vtResult*.

```
vtResult:=Delete string("Lamborghini";6;6) ` vtResult obtiene "Lambo"
vtResult:=Delete string("Indentation";6;2) ` vtResult obtiene "Indention"
vtResult:=Delete string(var;3;32000) ` vtResult es igual a los dos primeros caracteres de var
```

Get localized string

Get localized string (resNombre) -> Resultado

Parámetro	Tipo		Descripción
resNombre	Cadena	⇒	Nombre del atributo resNombre
Resultado	Cadena	⇐	Valor de la cadena designada por resNombre en el lenguaje actual

Descripción

El comando **Get localized string** devuelve el valor de la cadena designada por el atributo *resNombre* para el lenguaje actual.
Este comando sólo funciona dentro de una arquitectura XLIFF. Para mayor información sobre este tipo de arquitectura, por favor consulte la descripción de soporte de XLIFF en el *manual de Diseño*.
Nota: el comando **Get database localization** puede utilizarse para buscar el lenguaje utilizado por la aplicación.
Pase en *resNombre* el nombre del recurso de la cadena en la cual quiere obtener la traducción en el lenguaje objetivo actual.
Tenga encuenta que XLIFF es diacrítica.

Ejemplo

Este es un extracto del archivo .xlf:

```
<file source-language="en-US" target-language="es-ES"> [...]  <trans-unit resname="Show on disk">  <source>Show on disk</source>  <target>Mostrar en el disco</target>  </trans-unit>
```

Después de la ejecución de la siguiente instrucción:

```
$valorES:=Get localized string("Show on disk")
```

... si el lenguaje actual es el Español, $valorES contiene "Mostrar en el disco".

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable OK toma el valor 1. Si *resNombre* no se encuentra, el comando devuelve una cadena vacía y la variable OK toma el valor 0.

GET TEXT KEYWORDS

GET TEXT KEYWORDS (texto ; arrPalabrasClaves {; *})

Parámetro	Tipo		Descripción
texto	Texto	➡	Texto original
arrPalabrasClaves	Array texto	⬅	Array que contiene las palabras claves
*	Operador	➡	Si se pasa = palabras únicas

Descripción

El comando **GET TEXT KEYWORDS** divide todo el *texto* en palabras individuales y crea, para cada palabra obtenida, un elemento en el array texto *arrPalabrasClaves* para cada palabra.

4D utiliza el mismo algoritmo para dividir el texto en palabras individuales que utiliza para crear los **Índice de palabras claves**. Este algoritmo está basado en la librería ICU. Para más información sobre cómo se separa un texto en palabras, consulte la siguiente dirección: http://userguide.icu-project.org/boundaryanalysis.

Nota: por petición de los usuarios, se introdujo una excepción para el francés y el italiano: el apóstrofe (') seguido por una vocal o la letra "h" se considera como un separador de palabra. Por ejemplo, las cadenas "L'homme" o "l'arbre" se dividen en "L'"+"homme" y "l'"+"arbre".

El algoritmo utilizado difiere si la opción **Considerar sólo caracteres no alfanuméricos para las palabras claves** está seleccionada en las propiedades de la base (consulte **Página Base de datos/Almacenamiento de datos** en el manual de *Diseño*).

En el parámetro *texto*, pase el texto original a dividir en palabras. Este texto puede tener estilo, en cuyo caso las etiquetas de estilo se ignoran.

Para el parámetro *arrPalabrasClaves*, el comando llena este array texto con las palabras extraídas del texto.

Si pasa el parámetro opcional *, el comando sólo almacena cada palabra diferente una vez en *arrPalabrasClaves*. Por defecto, si este parámetro se omite, todas las palabras extraídas del texto se guardan en el array, incluso si aparecen varias veces.

Este comando permite efectuar de manera simple las búsquedas entre los registros que contienen grandes cantidades de texto garantizando utilizar las mismas palabras claves que 4D. Por ejemplo, imagine que tiene un texto que contiene "10,000 Jean-Pierre BC45". Si este texto se divide en las palabras claves "10,000" + "Jean-Pierre" + "BC45", el array contendrá 4 elementos. Entonces es fácil hacer un bucle en este array para encontrar los registros que contienen una o más de estas palabras clave utilizando el operador % (ver ejemplos).

Ejemplo 1

En un formulario que contiene un área de búsqueda, los usuarios pueden introducir una o más palabras. Cuando un usuario valida este formulario, buscamos los registros cuyo campo *Micampo* contenga al menos una de las palabras claves introducidas por el usuario.

```
 // vSearch es la variable del área de búsqueda en el formulario
GET TEXT KEYWORDS(vSearch;arrSearch;*)
 //* en el caso de que un usuario introduzca la misma palabra más de una vez
CREATE SET([MiTabla];"Totalfound")
$n:=Size of array(arrSearch)
For($i;1;$n)
   QUERY([MiTabla];[MiTabla]MiCampo % arrSearch{$i})
   CREATE SET([MiTabla];"encontrado")
   UNION("Totalfound";"encontrado";"Totalfound")
End for
USE SET("Totalfound")
```

Ejemplo 2

En el mismo formulario que antes, buscamos los registros donde el campo *MiCampo* contenga todas las palabras claves introducidas por el usuario.

```
 // vSearch es la variable del área de entrada en el formulario
GET TEXT KEYWORDS(vSearch;arrSearch;*)
$n:=Size of array(arrSearch)
QUERY([MiTabla];[MiTabla]MiCampo >=0;*)
 // inicializar la búsqueda = todos los registros
For($i;1;$n)
   QUERY([MyTable];&;[MyTable]MyField % arrSearch{$i};*)
 // añadir el criterio
```

```
End for
QUERY([MiTabla]) //búsqueda
```

Ejemplo 3

Para contar las palabras de un texto:

```
GET TEXT KEYWORDS(vText;arrWords) // todas las palabras
$n:=Size of array(arrWords)
GET TEXT KEYWORDS(vText;arrWords;*) // palabras diferentes
$m:=Size of array(arrWords)
ALERT("Este texto contiene "+String($n)+" palabras de "+String($m))
```

Insert string

Insert string (fuente ; ainsertar ; posicion) -> Resultado

Parámetro	Tipo		Descripción
fuente	Cadena	→	Cadena en la cual insertar otra cadena
ainsertar	Cadena	→	Cadena a insertar
posicion	Entero largo	→	Posición de la inserción
Resultado	Cadena	↩	Cadena resultante

Descripción

Insert string inserta la cadena de caracteres alfanuméricos *ainsertar* en la cadena *fuente* a partir de *posicion* y devuelve la cadena de caracteres resultante. La cadena *ainsertar* se coloca antes del carácter designado por *posicion*.

Si *ainsertar* es una cadena vacía (""), **Insert string** devuelve *fuente* sin cambios.

Si *posicion* es mayor a la longitud de *fuente*, *ainsertar* se añade al final de *fuente*. Si *posicion* es inferior a uno (1), *ainsertar* se inserta antes de *fuente*.

Insert string es diferente de **Change string** en que esta función inserta caracteres en lugar de reemplazarlos.

Ejemplo

El siguiente ejemplo ilustra el uso de **Insert string**. Los resultados se asignan a la variable *vtResult*.

```
vtResult:=Insert string("El verde";"árbol ";4)  ` vtResult obtiene "El árbol verde"
vtResult:=Insert string("Tala";"b";3)  ` vtResult es igual a "Tabla"
vtResult:=Insert string("Indenficación";"ti";6)  ` vtResult es igual a "Indentación"
```

Length

Length (cadena) -> Resultado

Parámetro	Tipo		Descripción
cadena	Cadena	➡	Cadena de la cual devolver la longitud
Resultado	Entero largo	➲	Longitud de la cadena

Descripción

Length permite obtener la longitud de *cadena*. **Length** devuelve el número de caracteres alfanuméricos en *cadena*.

Nota: en modo Unicode, si quiere verificar que una cadena contiene caracteres, incluyendo caracteres ignorables, debe utilizar el último If(Length(vtAnyText)=0) en lugar de If(vtAnyText=""). Si la cadena contiene por ejemplo Char(1), que es carácter ignorable, Length(vtAnyText) devuelve 1 pero vtAnyText="" devuelve True.

Ejemplo

Este ejemplo ilustra el uso de **Length**. Los resultados, descritos en los comentarios, se asignan a la variable *vlResult*.

```
vlResult:=Length("Topacio")  ` vlResult obtiene 7
vlResult:=Length("Ciudadano")  ` vlResult obtiene 9
```

Lowercase

Lowercase (laCadena {; *}) -> Resultado

Parámetro	Tipo		Descripción
laCadena	Cadena	➡	Cadena a pasar a minúsculas
*	Operador	➡	Si se pasa: conservar los acentos
Resultado	Cadena	⮌	Cadena en minúsculas

Descripción

Lowercase devuelve una cadena de caracteres igual a *laCadena* con todos los caracteres alfabéticos convertidos en minúsculas.
El parámetro opcional *, si se pasa, indica que los eventuales caracteres acentuados presentes en *laCadena* deben devolverse como caracteres en minúsculas con acentos. Por defecto, cuando se omite este parámetro, los caracteres acentuados "pierden" sus acentos después de que se lleva a cabo la conversión.

Ejemplo 1

El siguiente método de proyecto convierte a mayúsculas el primer carácter de la cadena o del texto recibido como parámetro. Por ejemplo, Nom:= Mayus ("juan") devuelve "Juan".

```
` Método de proyecto Mayus
` Mayus( Cadena) -> Cadena
` Mayus( Todo texto o cadena) -> texto con una letra en mayúsculas

$0:=Lowercase($1)
If(Length($0)>0)
   $0[[1]]:=Uppercase($0[[1]])
End if
```

Ejemplo 2

Este ejemplo compara los resultados obtenidos de acuerdo a si se pasa o no el parámetro *:

```
$lacadena:=Lowercase("DÉJÀ VU")  ` $lacadena es "deja vu"
$lacadena:=Lowercase("DÉJÀ VU";*)  ` $lacadena es "déjà vu"
```

Match regex

Match regex (patron ; laCadena ; inicio {; pos_encont. ; long_encont.}{; *}) -> Resultado

Parámetro	Tipo		Descripción
patron	Texto	➡	Expresión regular
laCadena	Texto	➡	Cadena en la cual se efectúa la búsqueda
inicio	Entero largo	➡	Posición de laCadena donde comenzar la búsqueda
pos_encont.	Array entero largo, Variable entero largo	⬅	Posición de la ocurrencia
long_encont.	Array entero largo, Variable entero largo	⬅	Longitud de la ocurrencia
*	Operador	➡	Si se pasa: buscar únicamente en la posición indicada
Resultado	Booleano	↩	True = la búsqueda ha encontrado una ocurrencia; De lo contrario, False.

Match regex (patron ; laCadena) -> Resultado

Parámetro	Tipo		Descripción
patron	Texto	➡	Expresión regular (igualdad completa)
laCadena	Texto	➡	Cadena en la cual se efectúa la búsqueda
Resultado	Booleano	↩	True= la cadena encuentra una ocurrencia, de lo contrario False

Descripción

El comando **Match regex** permite probar la conformidad de una cadena de caracteres con respecto a un conjunto de reglas sintetizadas por medio de un metalenguaje llamado "expresión regular" o "expresión racional." La abreviación regex es comúnmente empleada para indicar esto tipos de notaciones.

Pase en *patrón* la expresión regular a buscar. Consiste de un conjunto de caracteres utilizado para describir una cadena de caracteres, utilizando caracteres especiales.

Pase en *laCadena* la cadena en la cual buscar la expresión regular.

Pase en *inicio*, la posición en *laCadena* donde debe comenzar la búsqueda.

Si *pos_encont.*y *long_encont* son variables, el comando devuelve la posición y la longitud de la ocurrencia en estas variables. Si pasa arrays, el comando devuelve la posición y la longitud de la ocurrencia en el elemento cero de los arrays y las posiciones y longitudes de los grupos capturadas por la expresión regular en los elementos siguientes.

El parámetro opcional * indica, cuando se pasa, que la búsqueda debe llevarse a cabo en la posición especificada por *inicio* sin buscar más allá en caso de falla.

El comando devuelve **True** si la búsqueda encuentra una ocurrencia.

Para mayor información sobre regex, consulte la siguiente dirección:
http://en.wikipedia.org/wiki/Regular_expression

Para mayor información sobre la sintaxis de las expresiones regulares pasadas en el parámetro *patrón*, consulte la siguiente dirección:
http://www.icu-project.org/userguide/regexp.html

Ejemplo 1

Búsqueda de igualdad completa (sintaxis simple):

vencontrado:=Match regex(plantilla;mitexto)

```
QUERY BY FORMULA([Empleados];Match regex(".*smith.*";[Empleados]nombre))
```

Ejemplo 2

Búsqueda en el texto por posición:

```
vencont:=Match regex(motivo;mitexto;inicio;pos_encont;long_encont)
```

Ejemplo para mostrar todas las etiquetas de $1:

```
inicio:=1
Repeat
   vencont:=Match regex("<.*>";$1;inicio;pos_encont;long_encont)
   If(vencont)
      ALERT(Substring($1;</Gen9><span class="rte4d_prm">pos_encont</span><Gen9>;</Gen9><span
class="rte4d_prm">long_encont</span><Gen9>))
      inicio:=pos_encont+long_encont
   End if
Until(Not(vencont))
```

Ejemplo 3

Búsqueda con soporte de "grupos capturados" vía paréntesis. ( ) se utilizan para especificar grupos en los regex:
vencont:=Match regex(motivo;mitexto; inicio; pos_encont; long_encont_array)

```
ARRAY LONGINT(pos_encont_array;0)
ARRAY LONGINT(long_encont_array;0)
vencont:=Match regex("(.*)stuff(.*)";$1;1;pos_encont_array;long_encont_array)
If(vencont)
    $grupo1:=Substring($1;pos_encont_array{1};long_encont_array{1})
    $grupo2:=Substring($1;pos_encont_array{2};long_encont_array{2})
End if
```

Ejemplo 4

Búsqueda limitando la comparación del motivo a la posición indicada:
Añadir una estrella al final de una de las dos sintaxis anteriores.

```
vencont:=Match regex("a.b";"---a-b---";1;$pos_encont;$long_encont )
  `devuelve True
vencont:=Match regex("a.b";"---a-b---";1;$pos_encont;$long_encont ;*)
  `devuelve False
vencont:=Match regex("a.b";"---a-b---";4;$pos_encont;$long_encont ;*)
  `devuelve True
```

Nota: las posiciones y largos devueltos son significativosn sólo en modo Unicode o si el texto manipulado es de tipo ASCII 7-bits.

Gestión de errores

En caso de error, el comando genera un error que puede interceptar vía un método instalado por el comando **ON ERR CALL**.

Num

Num (expresion {; separador}) -> Resultado

Parámetro	Tipo		Descripción
expresion	Cadena, Booleano, Entero largo	➡	Cadena a convertir en numérico o Booleano a convertir en 0 ó 1, o Expresión numérica
separador	Cadena	➡	Separador decimal
Resultado	Real	↩	Valor numérico del parámetro expresión

Descripción

El comando **Num** devuelve en forma numérica la expresión de tipo cadena, booleano, o numérica que pasó en *expresion*. El parámetro opcional *separador* puede utilizarse para designar un separador decimal para la evaluación de las expresiones de tipo cadena.

Expresiones de tipo cadena

Si *expresion* consiste únicamente uno o más caracteres alfabéticos, **Num** devuelve cero. Si *expresion* incluye caracteres alfabéticos y caracteres numéricos, **Num** ignora los caracteres alfabéticos. De esta forma, **Num** transforma la cadena "a1b2c3" en el número 123.

Hay tres caracteres reservados que **Num** trata de manera particular: el separador decimal definido en el sistema (si el parámetro *separador* no se pasa), el guión "-", y "*e*" *o* "**E**". Estos caracteres son interpretados como caracteres de formato numérico.

- El separador decimal se interpreta como un lugar decimal y debe aparecer en una cadena numérica. Por defecto, el comando utiliza el separador decimal definido en el sistema operativo. Puede modificar este carácter utilizando el parámetro *separador* (ver a continuación).
- El guión define un número o exponente negativo. El guión debe aparecer antes de los caracteres numéricos negativos o después de la "e" para un exponente. Excepto por el carácter "e", si un guión está en una cadena numérica, la porción de la cadena después del guión se ignora. Por ejemplo, **Num**("123-456") devuelve 123, pero **Num**("-9") devuelve -9.
- La e o E hace que todos los caracteres numéricos a la derecha se interpreten como la potencia de un exponente. La "e" debe estar en una cadena numérica. De esta forma, **Num**("123e–2") devuelve 1,23.

Note que en el caso de que una cadena tenga más de un carácter "e", la conversión podrá dar resultados diferentes bajo Mac OS y bajo Windows.

El parámetro *separador* puede utilizarse para designar un separador decimal personalizado para la evaluación de *expresion*. Cuando la cadena a evaluar se expresa con un separador decimal diferente del separador del sistema, el comando devuelve un resultado incorrecto. El parámetro *separador* puede utilizarse en este caso para obtener una evaluación correcta. Cuando se pasa este parámetro, el comando no tiene en cuenta el separador decimal del sistema. Puede pasar uno o más caracteres.

Nota: el comando **GET SYSTEM FORMAT** puede utilizarse para buscar el separador decimal actual como también otros parámetros sistema regionales.

Expresiones de tipo Booleano

Si pasa una expresión booleana en el parámetro expresión, **Num** devuelve 1 si la expresión es True; de lo contrario devuelve *0* (cero).

Expresiones numéricas

Si pasa una expresión numérica en el parámetro *expresion*, **Num** devuelve tal cual el valor pasado en *expresion*. Esto puede ser útil especialmente en el caso de programación genérica utilizando punteros.

Expresiones indefinidas
Si la *expresion* se evalúa como indefinida, el comando devuelve 0 (cero). Esto es útil cuando se espera que el resultado de una expresión (por ejemplo, un atributo objeto) sea un número, incluso si puede ser indefinido.

Ejemplo 1

El siguiente ejemplo ilustra cómo funciona **Num** cuando se pasa un argumento de tipo cadena. Cada línea asigna un número a la variable *vResult*. Los comentarios describen los resultados:

```
vResult:=Num("ABCD")  ` vResult vale 0
vResult:=Num("A1B2C3")  ` vResult vale 123
vResult:=Num("123")  ` vResult vale 123
vResult:=Num("123,4")  ` vResult vale 123,4
vResult:=Num("-123")  ` vResult vale -123
vResult:=Num("-123e2")  ` vResult vale -12300
```

Ejemplo 2

En este ejemplo, *[Cliente]Deuda* se compara con el valor *$1000*. El comando **Num** aplicado a esta comparación devuelve 1 o 0. La multiplicación de una cadena por 1 ó 0 devuelve la cadena o la cadena vacía. Como resultado, *[Cliente]Riesgo* recibe el valor "Aceptable" o "Inaceptable":

```
// Si el cliente tiene deudas menores a 1000, el riesgo es aceptable.
// Si el cliente tiene deudas superiores a 1000, el riesgo es inaceptable.
[Cliente]Riesgo:=("Aceptable"*Num([Cliente]Deuda<1000))+("Inaceptable"*Num([Cliente]Deuda>=1000))
```

Ejemplo 3

Este ejemplo compara los resultados obtenidos dependiendo del separador "actual":

```
$lacadena:="33,333.33"
$elnum:=Num($lacadena)
  ` by default, $elnum es igual a 33,33333 en un sistema francés
$elnum:=Num($lacadena;".")
  ` $elnum se evaluará correctamente sin importar el sistema;
  ` por ejemplo, 33 333,33 en un sistema francés
```

Position

Position (aBuscar ; laCadena {; inicio {; longEncont}}{; *}) -> Resultado

Parámetro	Tipo		Descripción
aBuscar	Cadena	➡	Cadena a buscar
laCadena	Cadena	➡	Cadena en la cual buscar
inicio	Entero largo	➡	Posición en la cadena donde comenzar la búsqueda
longEncont	Entero largo	⬅	Longitud de la cadena encontrada
*	Operador	➡	Si se pasa: búsqueda diacrítica
Resultado	Entero largo	↩	Posición de la primera ocurrencia

Descripción

Position devuelve la posición de la primera ocurrencia de *aBuscar* en *laCadena*.

Si *laCadena* no contiene *buscar*, devuelve cero (0).

Si **Position** ubica una ocurrencia de *aBuscar*, la función devuelve la posición del primer carácter de esta ocurrencia en *laCadena*.

Si pregunta por la posición de una cadena vacía dentro de una cadena vacía, **Position** devuelve cero (0).

Por defecto, la búsqueda comienza en el primer carácter de *laCadena*. El parámetro opcional *inicio* permite precisar el carácter donde la búsqueda debe comenzar en *laCadena*.

El parámetro *longEncont*, si se pasa, devuelve la longitud de la cadena encontrada por la búsqueda. Este parámetro es necesario para poder gestionar correctamente cartas escritas con uno o más caracteres (ejemplo: æ y ae, ß y ss, etc.).
Tenga en cuenta que cuando se pasa el parámetro * (modo diacrítico, ver a continuación), estas letras no se consideran como equivalente (æ # ae); en modo diacrítico, *longEncont* siempre es igual a la longitud de *aBuscar* (si se encuentra una ocurrencia).
Por defecto, el comando hace comparaciones globales teniendo en cuenta particularidades lingüísticas y letras que pueden estar escritas con uno o más caracteres (por ejemplo æ = ae). Por otra parte, no es diacrítica (a=A, a=à etc.) y no tiene en cuenta los caracteres "ignorables". Los caracteres ignorables incluyen todos los caracteres del subset unicode *CO Control* (U+0000 a U+001F, ascii character control set) excepto los caracteres imprimibles (U+0009 TAB, U+0010 LF, U+0011 VT, U+0012 FF y U+0013 CR).
Para modificar este funcionamiento, pase asterisco * como último parámetro. En este caso, las comparaciones se basan en los códigos de los caracteres. Debe pasar el parámetro *:

- Si quiere tener en cuenta caracteres especiales, utilizados por ejemplo como delimitadores (**Char(1)**, etc.),
- Si la evaluación de caracteres debe ser sensible a las mayúsculas y minúsculas y tener en cuenta los caracteres acentuados (a # A, à # a, etc.)
  Note que en este modo, la evaluación no maneja variaciones en la escritura de las palabras.

Nota: en ciertos casos, utilizar el parámetro * puede acelerar significativamente la ejecución del comando.

Advertencia: no puede utilizar el carácter arroba @ con **Position**. Por ejemplo, si pasa *"abc@"* en *aBuscar*, el comando buscará la cadena *"abc@"* y no "abc" seguido de otros caracteres.

Ejemplo 1

Este ejemplo ilustra el uso de **Position**. Los resultados, descritos en los comentarios, se asignan a la variable *vlResult*.

```
vlResult:=Position("ll";"Billar")  ` vlResult toma el valor 3
vlResult:=Position(vtText1;vtText2)  ` Posición de la primera ocurrencia de vtText1 en vtText2
vlResult:=Position("todo";"todos los procesos dentro de un método";1)  ` vlResult toma el valor 1
vlResult:=Position("todo";"todos los procesos dentro de un método";2)  ` vlResult toma el valor 35
vlResult:=Position("TODO";"todos los procesos dentro de un método";1;*)  ` vlResult toma el valor 0
vlResult:=Position("œ";"Bœuf";1;$largo)  ` vlResult =2, $largo= 1
```

Ejemplo 2

En el siguiente ejemplo, el parámetro *longEncont* permite buscar todas las ocurrencias de "aegis" en un texto, sin importar cómo está escrito:

```
$inicio:=1
Repeat
   vlResult:=Position("aegis";$text;$inicio;$longEncont)
   $inicio:=$inicio+$longEncont
Until(vlResult=0)
```

Replace string

Replace string (fuente ; obsoleta ; nueva {; reemplazos}{; *}) -> Resultado

Parámetro	Tipo		Descripción
fuente	Cadena	➡	Cadena original
obsoleta	Cadena	➡	Caracteres a reemplazar
nueva	Cadena	➡	Cadena de reemplazo (si la cadena está vacía, se borran todas las ocurrencias)
reemplazos	Entero largo	➡	Número de reemplazos a efectuar Si se omite, se reemplazan todas las ocurrencias
*	Operador	➡	Si se pasa: evaluación basada en los códigos de los caracteres
Resultado	Cadena	↩	Cadena resultante

Descripción

Replace string devuelve una cadena de caracteres resultante de reemplazar *obsoleto* por *nuevo* en *fuente*.

Si *nuevo* es una cadena vacía (""), **Replace string** borra cada ocurrencia de *obsoleto* en *fuente*.

Si se especifica *reemplazos*, **Replace string** sólo reemplazará el número de ocurrencias especificado de *obsoleto*, a partir del primer carácter de *fuente*. Si no se especifica *reemplazos*, se reemplazan todas las ocurrencias de *obsoleto*.

Si *reemplazos* es una cadena vacía, **Replace string** devuelve *fuente* intacto.

Por defecto, el comando no tiene en cuenta si los caracteres están en mayúsculas o minúsculas o si están o no acentuados (a=A, a=à, etc.). Si pasa un asterisco * como último parámetro, indica que la evaluación de los caracteres debe ser diacrítica, en otras palabras, debe tener en cuenta las mayúsculas, minúsculas y caracteres acentuados (a#A, a#à...).

Por defecto, el comando hace comparaciones globales teniendo en cuenta particularidades lingüísticas y de las letras que pueden escribirse con uno o más caracteres (por ejemplo æ = ae). Por otra parte, no es diacrítico (a=A, a=à, etc.) y no tiene en cuenta los caracteres "ignorables" tales como los caracteres cuyo código es < 9 (especificación Unicode).

Para modificar este funcionamiento, pase como último parámetro *. En este caso, las comparaciones estarán basadas en códigos de caracteres. Debe pasar el parámetro *:

- Si quiere reemplazar los caracteres especiales, utilizados por ejemplo como delimitadores (**Char(1)**...),
- Si el reemplazo de caracteres debe tener en cuenta las mayúsculas, minúsculas y los caracteres acentuados (a#A, a#à, etc.).
  Note que en este modo, la evaluación no maneja variaciones de escritura de las palabras.

Nota: en 4D v15 R3 y superior, se hizo una optimización significativa al algoritmo utilizado por este comando cuando se reemplaza una cadena por otra de diferente longitud, independientemente de la sintaxis utilizada. Esto da como resultado una aceleración considerable del procesamiento en este contexto.

Ejemplo 1

El siguiente ejemplo ilustra el uso de **Replace string**. Los resultados, descritos en los comentarios, son asignados a la variable *vtResult*.

```
vtResult:=Replace string("Ventanilla";"illa";"d")  ` vtResult es igual a "Ventana"
vtResult:=Replace string("Ventanilla";"ill";"")  ` vtResultes igual a "Ventana"
vtResult:=Replace string(vtOtraVar;Char(Tab);",";*)  ` Reemplazar todas las tabulaciones en vtOtraVar por comas
```

Ejemplo 2

El siguiente ejemplo elimina los retornos de carro y las tabulaciones del texto en la variable *vtResult*:

```
vtResult:=Replace string(Replace string(vtResult;Char(Carriage return);"";*);Char(Tab);"")
```

Ejemplo 3

El siguiente ejemplo ilustra el uso del parámetro * en el caso de una evaluación diacrítica:

```
vtResult:=Replace string("Crème brûlée";"Brulee";"caramel")  `vtResult es igual a "Crème caramel"
vtResult:=Replace string("Crème brûlée";"Brulee";"caramel";*)  `vtResult es igual a "Crème brûlée"
```

Split string

Split string (cadenaASeparar ; separador {; opciones}) -> Resultado

Parámetro	Tipo		Descripción
cadenaASeparar	Texto	➡	Valor de la cadena
separador	Texto	➡	Cadena en la que cadenaASeparar se divide. Si cadena vacía (""), cada carácter de cadenaASeparar es una subcadena
opciones	Entero largo	➡	Opciones relativas a las cadenas vacías y espacios
Resultado	Collection	↩	Colección de subcadenas

Descripción

El comando **Split string** devuelve una colección de cadenas, creada al dividir *cadenaAseparar* en subcadenas en los límites especificados por el parámetro *separador*. Las subcadenas en la colección devuelta no incluyen el *separador*.

Si no se encuentra un *separador* en *cadenaAseparar*, **Split string** devuelve una colección que contiene un elemento único, *cadenaAseparar*. Si pasó una cadena vacía en *separador*, **Split string** devuelve una colección de cada carácter de *cadenaAseparar*.

En el parámetro *opciones*, puede pasar una o una combinación de las siguientes constantes del tema **Cadenas**:

Constante	Tipo	Valor	Comentario
sk ignore empty strings	Entero largo	1	Eliminar las cadenas vacías de la colección resultante (se ignoran)
sk trim spaces	Entero largo	2	Retirar los espacios al principio y al final de las subcadenas

Ejemplo 1

```
C_TEXT($vt)
C_COLLECTION($col)
$col:=New collection

$vt:="John;Doe;120 jefferson st.;Riverside;; NJ; 08075"
$col:=Split string($vt;";")  //["John","Doe","120 jefferson st.","Riverside","","," NJ","," 08075"]
$col:=Split string($vt;";";sk ignore empty strings)  //["John","Doe","120 jefferson st.","Riverside","," NJ","," 08075"]
$col:=Split string($vt;";";sk ignore empty strings+sk trim spaces)  //["John","Doe","120 jefferson st.","Riverside","NJ","08075"]
```

Ejemplo 2

El parámetro *separador* puede ser una cadena de múltiples caracteres:

```
C_TEXT($vt)
C_COLLECTION($col)
$vt:="Name<tab>Smith<tab>age<tab>40"
$col:=Split string($vt;"<tab>")
 //$col=["Name","Smith","age","40"]
```

String

String (expresion {; formato {; horaComb}}) -> Resultado

Parámetro	Tipo		Descripción
expresion	Expresión	➡	Expresión a convertir en cadena (puede ser de tipo Real, Entero, Entero largo, Fecha, Hora, Alfa, Texto o Booleana)
formato	Cadena, Entero largo	➡	Formato de visualización
horaComb	Hora	➡	Hora a combinar si expresión es una fecha
Resultado	Cadena	↩	Expresión convertida en cadena alfanumérica

Descripción

El comando **String** devuelve en forma de cadena alfanumérica la expresión de tipo numérico, Fecha, Hora, cadena o Booleana que se pasa en *expresion*.

Si no pasa el parámetro opcional *formato*, la cadena se devuelve en el formato por defecto del tipo de datos correspondiente. Si pasa *formato*, puede definir el formato de la cadena resultante.

El parámetro opcional *horaComb* añade un tiempo a una fecha en un formato combinado. Sólo puede utilizarse cuando el parámetro *expresion* es una fecha (ver a continuación).

Expresiones numéricas

Si *expresion* es una expresión numérica (Real, Entero, Entero largo), puede pasar el formato de la cadena opcional. Estos son algunos ejemplos:

Ejemplo	Resultado	Comentarios
String(2^15)	"32768"	Formato por defecto
String(2^15;"###,##0 habitantes")	"32,768 habitantes"	
String(1/3;"##0.00000")	"0.33333"	
String(1/3)	"0.3333333333333330000"	Formato por defecto(*)
String(Arctan(1)*4)	"3.141592653589790000"	Formato por defecto(*)
String(Arctan(1)*4;"##0.00")	"3.14"	
String(-1;"&x")	"0xFFFFFFFF"	
String(-1;"&$")	"$FFFFFFFF"	
String(0 ?+ 7;"&x")	"0x0080"	
String(0 ?+ 7;"&$")	"$80"	
String(0 ?+ 14;"&x")	"0x4000"	
String(0 ?+ 14;"&$")	"$4000"	
String(50,3;"&xml")	"50.3"	Siempre "." como un separador decimal
String(Num(1=1);"True;;False")	"True"	
String(Num(1=1);"True;;False")	"True"	
String(Num(1=2);"True;;False")	"False"	
String(Log(-1))	""	Número indefinido
String(1/0)	"INF"	Número infinito positivo
String(-1/0)	"-INF"	Número infinito positivo

(*) A partir de 4D v14 R3, el algoritmo de conversión de reales a texto está basado en 13 dígitos significativos (contrarío a los 15 dígitos en versiones anteriores de 4D).

El formato se especifica de la misma forma que para un campo numérico en un formulario. Para mayor información sobre formatos numéricos, consulte la sección **Formatos de salida** del manual de Diseño de 4D. Igualmente puede pasar el nombre de un estilo personalizado en *formato*. El nombre del estilo personalizado debe estar precedido por el carácter "|".

Nota: la función **String** no es compatible con campos de tipo "Entero 64 bits" en modo compilado.

Expresiones de tipo Fecha

Si *expresion* es de tipo Fecha, la cadena se devuelve en el formato por defecto definido en el sistema.
En el parámetro *formato*, puede pasar una de las constantes descritas a continuación (tema **Formatos de salida de fechas**):

En este caso, también puede pasar una hora en el parámetro *horaComb*. Este parámetro le permite combinar una fecha y una hora con el fin de generar marcadores de tiempo conforme a las normas vigentes (constantes ISO Date GMT y Date RFC 1123). Estos formatos son particularmente útiles en el contexto de los procesos XML y Web. El parámetro *horaComb* sólo se utiliza cuando el parámetro *expresion* es una fecha.

Constante	Tipo	Valor	Comentario
Blank if null date	Entero largo	100	"" en lugar de 0 en caso de valor nulo. Esta constante debe adicionarse al formato de visualización.
Date RFC 1123	Entero largo	10	Fri, 10 Sep 2010 13:07:20 GMT
Internal date abbreviated	Entero largo	6	29 dic, 2006
Internal date long	Entero largo	5	29 diciembre 2006
Internal date short	Entero largo	7	
Internal date short special	Entero largo	4	12/29/06 (pero 12/29/1896 o 12/29/2096)
ISO Date	Entero largo	8	2006-12-29T00:00:00 (obsoleto)
ISO Date GMT	Entero largo	9	2010-09-13T16:11:53Z
System date abbreviated	Entero largo	2	dom. 29 de 2006
System date long	Entero largo	3	domingo 29 diciembre 2006
System date short	Entero largo	1	

Nota: los formatos pueden variar dependiendo de la configuración del sistema.
Estos son algunos ejemplos de formatos simples (asumiendo que la fecha actual es 29/12/2009):

```
$vsResult:=String(Current date) ` $vsResult toma el valor "29/12/09"
$vsResult:=String(Current date;Internal date long) ` $vsResult toma el valor "29 de diciembre de 2009"
$vsResult:=String(Current date;ISO Date) ` $vsResult toma el valor "2009-12-29T00:00:00"
```

Notas para los formatos combinados fecha/hora:

- El formato ISO Date corresponde a la norma ISO8601. Este formato contiene una fecha y una hora. Por ejemplo, la fecha del 31 de mayo de 2006 a la 1:20 p.m. se escribe 2006-05-31T13:20:00. Si no pasa el parámetro *horaComb*, la parte hora se llena con 0 (ver ejemplo). Este formato expresa la fecha y hora local.

```
$mifecha:=String(Current date;ISO Date) // devuelve por ejemplo 2010-09-13T00:00:00
$mifecha:=String(Current date;ISO Date;Current time) // devuelve 2010-09-13T18:11:53
```

- El formato ISO Date GMT es similar al formato ISO Date, excepto que expresa la fecha y la hora con respecto a la zona del huso horario (hora GMT).

```
$mydate:=String(Current date;ISO Date GMT;Current time) // returns 2010-09-13T16:11:53Z
```

  Note que el caracter "Z" al final indica el formato GMT.
  Si solo pasa una fecha, el comando devuelve la fecha a la media noche (hora local) expresada en hora GMT lo cual puede causar que la fecha se mueva hacia adelante o hacia atrás dependiendo de la zona horaria local:

```
$mifecha:=String(Current date;ISO Date GMT) // devuelve 2010-09-12T22:00:00Z
```

- El formato Date RFC 1123 permite formatear un conjunto fecha/hora siguiendo la norma definida por los RFC 822 y 1123. Este formato es necesario por ejemplo para fijar la fecha de vencimiento de las cookies en un encabezado HTTP.

```
$mifecha:=String(Current date;Date RFC 1123;Current time) //devuelve, por ejemplo Fri, 10 Sep 2010 13:07:20 GMT
```

  La hora expresada tiene en cuenta la hora del huso horario (zona GMT). Si pasa una fecha, el comando devuelve la fecha a la media noche (hora loca) expresada en hora GMT lo que puede hacer que la fecha se mueva hacia adelante o hacia atrás dependiendo de la zona horaria local:

```
$mifecha:=String(Current date;Date RFC 1123) // devuelve Thu, 09 Sep 2010 22:00:00 GMT
```

Expresiones de tipo Hora

Si *expresion* es de tipo Hora, la cadena se devuelve utilizando el formato por defecto **HH:MM:SS**. Puede pasar en el parámetro *formato* una de las siguientes constantes del tema *"Time Display Formats"*):

Constante	Tipo	Valor	Comentario
Blank if null time	Entero largo	100	"" en lugar de 0
HH MM	Entero largo	2	
HH MM AM PM	Entero largo	5	
HH MM SS	Entero largo	1	
Hour min	Entero largo	4	1 hora 2 minutos
Hour min sec	Entero largo	3	1 hora 2 minutos 3 segundos
ISO time	Entero largo	8	
Min sec	Entero largo	7	62 minutos 3 segundos
MM SS	Entero largo	6	
System time long	Entero largo	11	1:02:03 AM HNEC (Mac únicamente)
System time long abbreviated	Entero largo	10	1•02•03 AM (Mac únicamente)
System time short	Entero largo	9	

Notas:

- El formato **ISO Date Time** corresponde a la norma ISO8601. Este formato contiene una fecha y una hora. Por ejemplo, la fecha del 31 de mayo de 2006 a la 1:20 p.m. se escribe 2006-05-31T13:20:00. Esto se utiliza para los procesos XML y con servicios Web. 4D no permite almacenar una fecha y hora en un solo campo. Sin embargo, es posible administrar las fechas en este formato utilizando el comando **String**.
- La constante **Blank if null** debe añadirse al formato; indica que en caso de un valor nulo 4D debe devolver una cadena vacía en lugar de ceros.

Estos ejemplos asumen que la hora actual 5:30 PM y 45 segundos:

```
$vsResult:=String(Current time) ` $vsResult toma el valor "17:30:45"
$vsResult:=String(Current time;Hour Min Sec) ` $vsResult toma el valor "17 horas 30 minutos 45 segundos"
```

Expresiones de tipo cadena

Si *expresion* es de tipo Alfa o Texto, el comando devuelve el mismo valor que se pasa en el parámetro. Esto puede ser útil particularmente en programación genérica utilizando punteros.
En este caso, si se pasa el parámetro *formato* se ignora.

Expresiones de tipo Booleano

Si *expresion* es de tipo Booleano, el comando devuelve la cadena “True” o “False” en el lenguaje de la aplicación (por ejemplo, “Vrai” o “Faux” en una versión francesa de 4D).

En este caso si se pasa el parámetro *formato*, se ignora.

Expresiones indefinidas

Si la *expresion* se evalúa como indefinida, el comando devuelve una cadena vacía. Esto es útil cuando se espera que el resultado de una expresión (por ejemplo, un atributo objeto) sea una cadena, incluso si puede ser indefinido.

Substring

Substring (fuente ; aPartirDe {; numCaracteres}) -> Resultado

Parámetro	Tipo		Descripción
fuente	Cadena	➡	Cadena de la cual obtener una subcadena
aPartirDe	Entero largo	➡	Posición del primer carácter
numCaracteres	Entero largo	➡	Número de caracteres a obtener
Resultado	Cadena	↩	Subcadena de fuente

Descripción

El comando **Substring** devuelve la parte de *fuente* definida por *aPartirDe* y *numCars*.

El parámetro *aPartirDe* indica el primer carácter de la cadena a devolver, y *numCars* define el número de caracteres a devolver.

Si *aPartirDe* más *numCars* es mayor que el número de caracteres en la cadena o si *numCars* no está especificado, **Substring** devuelve todos los caracteres de la cadena a partir del carácter especificado por *aPartirDe*. Si *aPartirDe* es superior al número de caracteres en la cadena, **Substring** devuelve una cadena vacía ("").

Atención: cuando se utiliza este comando en un contexto multi-estilo, debe convertir los eventuales caracteres de fin de línea Windows ('\r\n') en caracteres de fin de línea simples ('\r') para que el procesamiento sea válido. Esto se debe al mecanismo que normaliza los finales de línea 4D para asegurar la compatibilidad multi-plataforma para los textos. Para obtener más información, consulte **Normalización automática de fines de líneas**.

Ejemplo 1

Este ejemplo ilustra la utilización de **Substring**. Los resultados se asignan a la variable *vsResult*.

```
vsResult:=Substring("08/04/62";4;2) ` vsResult toma el valor "04"
vsResult:=Substring("Emergencia";1;6) ` vsResult toma el valor "Emerge"
vsResult:=Substring(var;2) ` vsResult toma el valor de todos los caracteres excepto el primero
```

Ejemplo 2

El siguiente método de proyecto añade los párrafos que se encuentran en el texto (pasado como primer parámetro) a una array de tipo texto o alfa (cuyo puntero se pasa como segundo parámetro):

```
  ` EXTRACT PARAGRAPHS
  ` EXTRACT PARAGRAPHS ( texto ; Puntero )
  ` EXTRACT PARAGRAPHS ( Texto a analizar ; -> Array de párrafos )

C_TEXT($1)
C_POINTER($2)

$vlElem:=Size of array($2->)
Repeat
   $vlElem:=$vlElem+1
   INSERT IN ARRAY($2->;$vlElem)
   $vlPos:=Position(Char(Carriage return);$1)
   If($vlPos>0)
      $2->{$vlElem}:=Substring($1;1;$vlPos-1)
      $1:=Substring($1;$vlPos+1)
   Else
      $2->{$vlElem}:=$1
   End if
Until($1="")
```

Uppercase

Uppercase (laCadena {; *}) -> Resultado

Parámetro	Tipo		Descripción
laCadena	Cadena	→	Cadena a convertir en mayúsculas
*	Operador	→	Si se pasa: conservar los acentos
Resultado	Cadena	←	Cadena en mayúsculas

Descripción

Uppercase devuelve una cadena de caracteres igual a *laCadena* con todos los caracteres alfabéticos convertidos en mayúsculas.
El parámetro opcional *, si se pasa, indica que los eventuales caracteres acentuados presentes en *laCadena* deben ser devueltos como caracteres en mayúsculas acentuados. Por defecto, cuando se omite este parámetro, los caracteres acentuados "pierden" sus acentos después de efectuada la conversión.

Ejemplo 1

Este ejemplo compara los resultados obtenidos según se pase o no el parámetro *:

```
$lacadena:=Uppercase("andrés") ` $lacadena es igual a "ANDRES"
$lacadena:=Uppercase("andrés";*) ` $lacadena es igual a "ANDRÉS"
```

Ejemplo 2

Ver el ejemplo para **Lowercase**.

_o_Convert case

_o_Convert case
Este comando no requiere parámetros

Descripción

Este comando es obsoleto y no debe ser utilizado.

_o_ISO to Mac

_o_ISO to Mac (texto) -> Resultado

Parámetro	Tipo		Descripción
texto	Cadena	➡	Texto expresado en juego de caracteres Web estándar
Resultado	Cadena	⮌	Texto expresado en ASCII Mac OS

Nota de compatibilidad

En modo Unicode, este comando no hace nada (la cadena texto se devuelve sin modificaciones). Desde la versión 11 de 4D, este comando es obsoleto y su uso no es recomendable. Se recomienda convertir las cadenas de caracteres utilizando los comandos **CONVERT FROM TEXT** o **Convert to text**.

_o_Mac to ISO

_o_Mac to ISO (texto) -> Resultado

Parámetro	Tipo		Descripción
texto	Cadena	➡	Texto en ASCII Mac OS
Resultado	Cadena	⮌	Texto en conjunto de caracteres estándar Web

Nota de compatibilidad

En modo Unicode, este comando no tiene efecto (la cadena *texto* se devuelve sin modificación). Desde la versión 11 de 4D, este comando es obsoleto y su uso no es recomendable. Se aconseja convertir las cadenas de caracteres utilizando los comandos **CONVERT FROM TEXT** o **Convert to text**.

_o_Mac to Win

_o_Mac to Win (texto) -> Resultado

Parámetro	Tipo		Descripción
texto	Cadena	➡	Texto expresado en ASCII Mac OS
Resultado	Cadena	⮌	Texto expresado en ANSI Windows

Nota de compatibilidad

En modo Unicode, este comando no tiene efecto (la cadena *texto* se devuelve sin modificaciones). Desde la versión 11 de 4D, este comando es obsoleto y su uso no es recomendable. Se aconseja convertir las cadenas de caracteres utilizando los comandos **CONVERT FROM TEXT** o **Convert to text**.

_o_Win to Mac

_o_Win to Mac (texto) -> Resultado

Parámetro	Tipo		Descripción
texto	Cadena	➡	Texto en ANSI Windows
Resultado	Cadena	⮌	Texto en ASCII Mac OS

Nota de compatibilidad

En modo Unicode, este comando no tiene efecto (la cadena *texto* se devuelve sin modificación). Desde la versión 11 de 4D, este comando es obsoleto y su uso no es recomendable. Se aconseja convertir las cadenas de caracteres utilizando los comandos **Convert to text** o **CONVERT FROM TEXT**.

Cliente HTTP

- HTTP AUTHENTICATE
- HTTP Get
- HTTP Get certificates folder
- HTTP GET OPTION
- HTTP Request
- HTTP SET CERTIFICATES FOLDER
- HTTP SET OPTION

HTTP AUTHENTICATE

HTTP AUTHENTICATE (nombre ; clave {; metodoAut} {; *})

Parámetro	Tipo		Descripción
nombre	Texto	➡	Nombre de usuario
clave	Texto	➡	Clave de usuario
metodoAut	Entero largo	➡	Método de autenticación: 0 o se omite = no definido, 1 = BASIC, 2 = DIGEST
*	Operador	➡	Si se pasa: autenticación por proxy

Descripción

El comando **HTTP AUTHENTICATE** permite efectuar peticiones HTTP a los servidores que necesitan la autenticación de la aplicación cliente. Los métodos BASIC y DIGEST son compatibles, así como también la presencia de un proxy.

En los parámetros *nombre* y *clave*, pase la información de identificación requerida(nombre de usuario y contraseña). Esta información se codifica y añade a la siguiente petición HTTP enviada utilizando el comando **HTTP Request** o **HTTP Get**, de manera que es necesario llamar al comando **HTTP AUTHENTICATE** antes de cada petición HTTP.

El parámetro opcional *metodoAut* permite indicar el método de autenticación a utilizar. Puede pasar una de las siguientes constantes, del tema **HTTP Client**:

Constante	Tipo	Valor	Comentario
HTTP basic	Entero largo	1	Utilizar el método de autenticación BASIC
HTTP digest	Entero largo	2	Utilizar el método de autenticación DIGEST

Si omite el parámetro *metodoAut* (o pasa 0), deja que el programa elija el método apropiado a utilizar. En este caso, 4D envía una petición adicional con el fin de negociar el método de autenticación.

Si pasa el parámetro *, indica que la información de autenticación se dirige a un proxy HTTP. Esta configuración debe implementarse cuando hay un proxy que requiere autenticación entre el cliente y el servidor HTTP. Si el servidor mismo se autentica, es necesaria una autenticación doble.

Por defecto, la información de autenticación se conserva y reutiliza en el proceso actual. Sin embargo, es posible reiniciar esta información utilizando una opción del comando **HTTP SET OPTION**. En este caso, será necesario ejecutar el comando **HTTP AUTHENTICATE** antes de cada llamada a **HTTP Request** o **HTTP Get**.

Ejemplo

Ejemplos de peticiones con autenticación:

```
// Autenticación en un servidor HTTP en modo DIGEST
HTTP AUTHENTICATE("httpUser";"123";2)
// Autenticación en un proxy en modo por defecto
HTTP AUTHENTICATE("ProxyUser";"456";*)
$httpStatus:=HTTP Get(...)
```

HTTP Get

HTTP Get (url ; respuesta {; nomEncab ; valoresEncab}{; *}) -> Resultado

Parámetro	Tipo		Descripción
url	Texto	➡	URL al cual enviar la petición
respuesta	Texto, BLOB, Imagen, Objeto	⬅	Resultado de la petición
nomEncab	Array texto	➡	Nombres de los encabezados de la petición
		⬅	Nombres de encabezados devueltos
valoresEncab	Array texto	➡	Valores de los encabezados de la petición
		⬅	Valores de los encabezados devueltos
*	Operador	➡	Si se pasa, la conexión se mantiene(keep-alive) Si se omite, la conexión se cierra automáticamente
Resultado	Entero largo	↩	Código de estado HTTP

Descripción

El comando **HTTP Get** envía directamente una petición HTTP GET a un URL específico y procesa la respuesta del servidor HTTP.
Pase en el parámetro *url* el URL al cual enviar la petición. La sintaxis a utilizar es:

```
http://[{user}:[{password}]@]host[:{port}][/{path}][?{queryString}]
```

Por ejemplo, puede pasar las siguientes cadenas:

```
http://www.myserver.com
http://www.myserver.com/path
http://www.myserver.com/path?name="jones"
https://www.myserver.com/login (*)
http://123.45.67.89:8083
http://john:smith@123.45.67.89:8083
http://[2001:0db8:0000:0000:0000:ff00:0042:8329]
http://[2001:0db8:0000:0000:0000:ff00:0042:8329]:8080/index.html (**)
```

(*) Durante las peticiones HTTPS, la autoridad del certificado no se verifica.
(**) Para más información sobre las direcciones IPv6 en urls, consulte RFC 2732.

Después de la ejecución del comando, el parámetro *respuesta* recupera el resultado de la petición devuelto por el servidor. Este resultado corresponde al cuerpo (body) de la respuesta, sin los encabezados (headers).
Puede pasar variables de diferentes tipos en *respuesta*:

- Texto: cuando el resultado se espera en forma de texto (ver nota abajo)
- BLOB: cuando el resultado se espera en forma binaria.
- Imagen: cuando el resultado se espera en forma de imagen.
- Objeto: cuando el resultado se espera en forma de objeto **C_OBJECT**.

Nota: cuando se pasa una variable de texto en *respuesta*, 4D intentará decodificar los datos devueltos desde el servidor. 4D primero intenta recuperar el conjunto de caracteres del encabezado de *tipo de contenido*, luego del contenido utilizando un BOM y, finalmente, busca cualquier atributo *http-equiv charset* (en contenido html) o *codificación* (para xml). Si no se puede detectar ningún charset, 4D intentará decodificar la respuesta en ANSI. Si la conversión falla, el texto resultante quedará vacío. Si no está seguro de si el servidor devuelve una información charset o BOM, pero conoce la codificación, es más preciso pasar *respuesta* en BLOB y llamar al **Convert to text**.
Si pasa un BLOB, contendrá el texto, la imagen o todo tipo de contenido (.wav, .zip, etc.) devuelto por el servidor. A continuación, debe gestionar la recuperación de estos contenidos (los encabezados no están incluidos en el BLOB). Si pasa un objeto de tipo **C_OBJECT**, si la petición devuelve un resultado con el contenido tipo "aplicación/json" (o "*algo*/json"), 4D intenta analizar el contenido JSON para generar el objeto.

En *nomEncab* y *valoresEncab* pase los arrays que contienen los nombres y los valores de los encabezados de la petición. Después de la ejecución del método, estos arrays contienen los nombres y los valores de los encabezados devueltos por el servidor HTTP. Más específicamente, este principio le permite administrar sus cookies.

El parámetro * permite activar el mecanismo keep-alive para la conexión al servidor. Por defecto, este parámetro se omite, keep-alive no está activo.

El comando devuelve el código del estado HTTP estándar (200=OK...) tal como fue devuelto por el servidor. La lista de códigos de estado HTTP está en el RFC 2616.
Si la conexión al servidor no es posible por una razón relacionada con la red (DNS Failed, Server not reachable...), el comando devuelve 0 y se genera un error. Puede interceptar estos errores utilizando un método de gestión de errores instalado por el comando **ON ERR CALL**.

Ejemplo 1

Recuperación del logo 4D en el sitio web de 4D:

```
C_TEXT(URLPic_t)
URLPic_t:="http://www.4d.com/sites/all/themes/dimention/images/home/logo4D.jpg"
```

```
ARRAY TEXT(HeaderNames_at;0)
ARRAY TEXT(HeaderValues_at;0)
C_PICTURE(Pic_i)
$httpResponse:=HTTP Get(URLPic_t;Pic_i;HeaderNames_at;HeaderValues_at)
```

Ejemplo 2

Recuperación de un RFC:

```
C_TEXT(URLText_t)
C_TEXT(Text_t)
URLText_t:="http://tools.ietf.org/rfc/rfc1.txt"
ARRAY TEXT(HeaderNames_at;0)
ARRAY TEXT(HeaderValues_at;0)
$httpResponse:=HTTP Get(URLText_t;Text_t;HeaderNames_at;HeaderValues_at)
```

Ejemplo 3

Recuperación de un vídeo:

```
C_BLOB(vBlob)
$httpResponse:=HTTP Get("http://www.example.com/video.flv";vBlob)
BLOB TO DOCUMENT("video.flv";vBlob)
```

HTTP Get certificates folder

HTTP Get certificates folder -> Resultado

Parámetro	Tipo		Descripción
Resultado	Texto		Ruta completa de la carpeta de certificados activa

Descripción

El comando **HTTP Get certificates folder** devuelve la ruta completa de la carpeta de certificados activa del cliente.

Ejemplo

Usted quiere cambiar temporalmente la carpeta de certificados:

```
C_TEXT($certifFolder)
$certifFolder :=HTTP Get certificates folder  //guardar carpeta actual
HTTP SET CERTIFICATES FOLDER("C:/temp/certifTempo/")
... // ejecución de peticiones específicas
HTTP SET CERTIFICATES FOLDER($certifFolder)  //restablecer la carpeta anterior
```

HTTP GET OPTION

HTTP GET OPTION (opción ; valor)

Parámetro	Tipo		Descripción
opción	Entero largo	➡	Código de la opción a leer
valor	Entero largo	⬅	Valor actual de la opción

Descripción

El comando **HTTP GET OPTION** devuelve el valor actual de las opciones HTTP (opciones utilizadas por el cliente para la próxima petición provocada por el comando **HTTP Get** o **HTTP Request**). El valor actual de una opción puede ser el valor por defecto o puede haber sido modificado utilizando el comando **HTTP SET OPTION**.

Nota: las opciones son locales al proceso actual. En un componente, son locales al componente en ejecución.

Pase en el parámetro *opcion* el número de la opción cuyo valor quiere leer. Puede utilizar una de las siguientes constantes predefinidas, disponibles en el tema **HTTP Client**:

Constante	Tipo	Valor	Comentario
HTTP compression	Entero largo	6	*valor* = 0 (no comprimir) ó 1 (comprimir). Por defecto: 0 Esta opción activa o desactiva el mecanismo de compresión de las peticiones entre el cliente y el servidor, para acelerar los intercambios. Cuando este mecanismo está activo, el cliente HTTP utiliza la compresión deflate o GZIP en función de la respuesta del servidor.
HTTP display auth dial	Entero largo	4	*valor* = 0 (no mostrar el diálogo) o 1 (mostrar el diálogo). Por defecto: 0 Esta opción controla la visualización de la caja de diálogo de autenticación al ejecutar el comando **HTTP Get** o **HTTP Request**. Por defecto, este comando no provoca la visualización de la caja de diálogo, normalmente debe utilizar el comando **HTTP AUTHENTICATE**. Sin embargo, si desea que aparezca una caja de diálogo de autenticación del usuario para que introduzca su nombre de usuario y contraseña, pase 1 en *valor*. La caja de diálogo aparece sólo si la solicitud requiere autenticación.
HTTP follow redirect	Entero largo	2	*valor* = 0 (no acepta redirecciones) o 1 (acepta redirecciones). Valor por defecto = 2
HTTP max redirect	Entero largo	3	*valor* = número máximo de redirecciones aceptadas Valor por defecto = 2
HTTP reset auth settings	Entero largo	5	*valor* = 0 (no borrar la información) ó 1 (borrar). Por defecto: 0 Esta opción permite indicar a 4D memorizar la información de autenticación del usuario (nombre de usuario, contraseña, método, etc.) Con el fin de volver a usarlos más adelante. Por defecto, esta información se borra después de cada ejecución del comando **HTTP Get** o **HTTP Request**. Pase 0 en *valor* para memorizarlos y para borrarlos. Tenga en cuenta que cuando se pasa 0, la información se conserva durante la sesión, pero no se guarda.
HTTP timeout	Entero largo	1	*valor* = timeout de la petición del cliente, expresada en segundos. El time out es el tiempo de espera del cliente HTTP en caso de no respuesta por parte del servidor. Al final de este período, el cliente cierra la sesión, la petición se pierde. Por defecto, este tiempo es de 120 segundos. Puede cambiarse en función de características específicas (estado de la red, características de la aplicación, etc).

En el parámetro *valor*, pase una variable para recibir el valor actual de la *opcion*.

HTTP Request

HTTP Request (metodoHTTP ; url ; contenido ; respuesta {; nomEncab ; valoresEncab}{; *}) -> Resultado

Parámetro	Tipo		Descripción
metodoHTTP	Texto	➡	Método HTTP para la petición
url	Texto	➡	URL a la cual enviar la petición
contenido	Texto, BLOB, Imagen, Objeto	➡	Contenido del cuerpo(body)de la petición
respuesta	Texto, BLOB, Imagen, Objeto	⬅	Resultado de la petición
nomEncab	Array texto	➡	Nombres de los encabezados de la petición
		⬅	Nombres de los encabezados devueltos
valoresEncab	Array texto	➡	Valores de los encabezados de la petición
		⬅	Valores de los encabezados devueltos
*	Operador	➡	Si se pasa, la conexión se mantiene (keep-alive) Si se omite, la conexión se cierra automáticamente
Resultado	Entero largo	➲	Código de estado HTTP

Descripción

El comando **HTTP Request** permite enviar todo tipo de petición HTTP a un URL específico y procesar la respuesta del servidor HTTP.

Pase en el parámetro *metodoHTTP* el método HTTP de la petición. Puede utilizar una de las siguientes constantes, del tema **HTTP Client**:

Constante	Tipo	Valor	Comentario
HTTP DELETE method	Cadena	DELETE	Ver el RFC 2616
HTTP GET method	Cadena	GET	Ver el RFC 2616. Equivale a utilizar el comando **HTTP Get**
HTTP HEAD method	Cadena	HEAD	Ver el RFC 2616
HTTP OPTIONS method	Cadena	OPTIONS	Ver el RFC 2616
HTTP POST method	Cadena	POST	Ver el RFC 2616
HTTP PUT method	Cadena	PUT	Ver el RFC 2616
HTTP TRACE method	Cadena	TRACE	Ver el RFC 2616

Pase en el parámetro *url* el URL a donde quiere enviar la petición. La sintaxis a utilizar es:

```
http://[{user}:[{password}]@]host[:{port}][/{path}][?{queryString}]
```

Por ejemplo, puede pasar las siguientes cadenas:

```
http://www.myserver.com
http://www.myserver.com/path
http://www.myserver.com/path?name="jones"
https://www.myserver.com/login (*)
http://123.45.67.89:8083
http://john:smith@123.45.67.89:8083
http://john:smith@123.45.67.89:8083
http://[2001:0db8:0000:0000:0000:ff00:0042:8329]
http://[2001:0db8:0000:0000:0000:ff00:0042:8329]:8080/index.html (**)
```

(*) Durante las peticiones HTTPS, la autoridad del certificado no se verifica.
(**) Para mayor información sobre las direcciones IPv6 en urls, por favor consulte RFC 2732.

Pase en el parámetro *contenido* el cuerpo (body) de la petición. Los datos pasados en este parámetro dependen del método HTTP de la petición.
Puede enviar datos de tipo texto, BLOB, imagen u objeto. Cuando el content-type no se especifica, se utilizan los siguientes tipos:

- para los textos: texto/plano - UTF8
- para los BLOBs: aplicación/byte-stream
- para las imágenes: tipo MIME conocido (best for Web).
- para los objetos **C_OBJECT**: aplicación/json

Después de la ejecución del comando, el parámetro *respuesta* recupera el resultado de la petición devuelto por el servidor. Este resultado corresponde al cuerpo (body) de la respuesta, sin los encabezados (headers). Puede pasar variables de diferentes tipos en *respuesta*:

- Texto: cuando el resultado se espera en forma de texto (ver nota abajo).
- BLOB: cuando el resultado se espera en forma binaria.
- Imagen: cuando el resultado se espera en forma de imagen.
- Objeto **C_OBJECT**: cuando el resultado esperado es un objeto.

Nota: cuando se pasa una variable de texto en *respuesta*, 4D intentará decodificar los datos devueltos desde el servidor. 4D primero intenta recuperar el conjunto de caracteres del encabezado de *tipo de contenido*, luego del contenido utilizando un BOM y, finalmente, busca cualquier atributo *http-equiv charset* (en contenido html) o *codificación* (para xml). Si no se puede detectar ningún charset, 4D intentará decodificar la respuesta en ANSI. Si la conversión falla, el texto resultante quedará vacío. Si no está

seguro de si el servidor devuelve una información charset o BOM, pero conoce la codificación, es más preciso pasar *respuesta* en BLOB y llamar al **Convert to text**.

Si pasa una variable de tipo **C_OBJECT** en el parámetro *respuesta*, si la petición devuelve un resultado con el contenido tipo "aplicación/json" (o "algo/json"), 4D intenta analizar el contenido JSON para generar el objeto.

Si el resultado devuelto por el servidor no corresponde al tipo de la variable *respuesta*, se deja vacío y la variable sistema OK toma el valor 0.

En *nomEncab* y *valoresEncab* pase los arrays que contienen los nombres y los valores de los encabezados de la petición. Después de la ejecución del método, estos arrays contienen los nombres y los valores de los encabezados devueltos por el servidor HTTP. Más específicamente, este principio le permite administrar sus cookies.

El parámetro * permite activar el mecanismo keep-alive para la conexión al servidor. Por defecto, este parámetro se omite, keep-alive no está activo.

El comando devuelve el código del estado HTTP estándar (200=OK...) tal como fue devuelto por el servidor. La lista de códigos de estado HTTP está en el RFC 2616.
Si la conexión al servidor no es posible por una razón relacionada con la red (DNS Failed, Server not reachable...), el comando devuelve 0 y se genera un error. Puede interceptar estos errores utilizando un método instalado por el comando **ON ERR CALL**.

Ejemplo 1

Solicitud de eliminación de un registro en una base remota:

```
C_TEXT($response)
$body_t:="{record_id:25}"
$httpStatus_l:=HTTP Request(HTTP DELETE method;"database.example.com";$body_t;$response)
```

Nota: usted debe procesar la solicitud de la manera apropiada en el servidor remoto, **HTTP Request** sólo se encarga de la petición y del resultado devuelto.

Ejemplo 2

Solicitud de adición de un registro a una base remota:

```
C_TEXT($response)
$body_t:="{fName:'john',fName:'Doe'}"
$httpStatus_l:=HTTP Request(HTTP PUT method;"database.example.com";$body_t;$response)
```

Nota: usted debe procesar la solicitud de la manera apropiada en el servidor remoto, **HTTP Request** sólo se encarga de la petición y del resultado devuelto.

Ejemplo 3

Petición para añadir un registro en JSON a una base remota:

```
C_OBJECT($content)
OB SET($content;"lastname";"Doe";"firstname";"John")
$result:=HTTP Request(HTTP PUT method;"database.example.com";$content;$response)
```

HTTP SET CERTIFICATES FOLDER

HTTP SET CERTIFICATES FOLDER (carpetaCertificados)

Parámetro	Tipo		Descripción
carpetaCertificados	Texto	➡	Ruta y nombre de la carpeta de certificados del cliente

Descripción

El comando **HTTP SET CERTIFICATES FOLDER** permite modificar la carpeta de certificados cliente activa para el conjunto de los procesos en la sesión actual.

La carpeta de certificados cliente es en la cual 4D busca los archivos de certificados cliente que son requeridos por los servidores web. Por defecto, siempre y cuando el comando **HTTP SET CERTIFICATES FOLDER** no se ejecute, 4D utiliza una carpeta llamada "ClientCertificatesFolder " que se crea junto al archivo de estructura. Esta carpeta se crea únicamente cuando es necesario.

En 4D v14, ahora es posible utilizar varios certificados clientes.

En *carpetaCertificados*, pase la ruta de acceso de la carpeta personalizada que contiene los certificados clientes. Puede pasar una ruta de acceso relativa al archivo de estructura de la aplicación, o una ruta de acceso absoluta. La ruta debe ser expresada con la sintaxis del sistema, por ejemplo:

- (OS X): Disk:Applications:myserv:folder
- (Windows): C:\Applications\myserv\folder

Una vez ejecutado este comando, la nueva ruta se tiene en cuenta inmediatamente por comandos tales como **HTTP Request** que se ejecuten después (no es necesario reiniciar la aplicación). Se utiliza en todos los procesos de la base.

Si la carpeta especificada no existe en la ubicación definida, o si la ruta de acceso pasada en *carpetaCertificados* no es válida, se genera un error. Puede interceptar este error utilizando un método de gestión de errores instalado por el comando **ON ERR CALL**.

Certificados SSL

Como se describe en la sección **Utilizar el protocolo TLS (HTTPS)**, los certificados SSL gestionados por 4D 4D deben estar en **PEM format**. Si su proveedor de certificados (por ejemplo, startssl) le envía un certificado que está en un formato binario como .crt, .pfx o .p12 (el formato también depende de su navegador), tiene que convertirlo al formato PEM para utilizarlo. Hay sitios web como sslshopper donde puede hacer esta conversión en línea.

Ejemplo

Usted quiere cambiar temporalmente la carpeta de certificados:

```
C_TEXT($certifFolder)
$certifFolder :=HTTP Get certificates folder  //guardar carpeta actual
HTTP SET CERTIFICATES FOLDER("C:/temp/certifTempo/")
... // ejecución de peticiones específicas
HTTP SET CERTIFICATES FOLDER($certifFolder)  //restablecer la carpeta anterior
```

HTTP SET OPTION

HTTP SET OPTION (opcion ; valor)

Parámetro	Tipo		Descripción
opcion	Entero largo	➡	Código de la opción a definir
valor	Entero largo	➡	Valor de la opción

Descripción

El comando **HTTP SET OPTION** permite definir diferentes opciones utilizadas durante la próxima petición disparada por los comandos **HTTP Get** o **HTTP Request**. Puede llamar este comando tantas veces como opciones a definir.

Nota: las opciones definidas son locales al proceso actual. Para componentes, son locales al componente en ejecución.

Pase en el parámetro *opcion* el número de la opción a definir y en el parámetro *valor* el nuevo valor de esta opción. Puede utilizar para el parámetro *opcion* una de las siguientes constantes, que se encuentran en el tema **HTTP Client**:

Constante	Tipo	Valor	Comentario
HTTP compression	Entero largo	6	*valor* = 0 (no comprimir) ó 1 (comprimir). Por defecto: 0 Esta opción activa o desactiva el mecanismo de compresión de las peticiones entre el cliente y el servidor, para acelerar los intercambios. Cuando este mecanismo está activo, el cliente HTTP utiliza la compresión deflate o GZIP en función de la respuesta del servidor.
HTTP display auth dial	Entero largo	4	*valor* = 0 (no mostrar el diálogo) o 1 (mostrar el diálogo). Por defecto: 0 Esta opción controla la visualización de la caja de diálogo de autenticación al ejecutar el comando **HTTP Get** o **HTTP Request**. Por defecto, este comando no provoca la visualización de la caja de diálogo, normalmente debe utilizar el comando **HTTP AUTHENTICATE**. Sin embargo, si desea que aparezca una caja de diálogo de autenticación del usuario para que introduzca su nombre de usuario y contraseña, pase 1 en *valor*. La caja de diálogo aparece sólo si la solicitud requiere autenticación.
HTTP follow redirect	Entero largo	2	*valor* = 0 (no acepta redirecciones) o 1 (acepta redirecciones). Valor por defecto = 2
HTTP max redirect	Entero largo	3	*valor* = número máximo de redirecciones aceptadas Valor por defecto = 2
HTTP reset auth settings	Entero largo	5	*valor* = 0 (no borrar la información) ó 1 (borrar). Por defecto: 0 Esta opción permite indicar a 4D memorizar la información de autenticación del usuario (nombre de usuario, contraseña, método, etc.) Con el fin de volver a usarlos más adelante. Por defecto, esta información se borra después de cada ejecución del comando **HTTP Get** o **HTTP Request**. Pase 0 en *valor* para memorizarlos y para borrarlos. Tenga en cuenta que cuando se pasa 0, la información se conserva durante la sesión, pero no se guarda.
HTTP timeout	Entero largo	1	*valor* = timeout de la petición del cliente, expresada en segundos. El time out es el tiempo de espera del cliente HTTP en caso de no respuesta por parte del servidor. Al final de este período, el cliente cierra la sesión, la petición se pierde. Por defecto, este tiempo es de 120 segundos. Puede cambiarse en función de características específicas (estado de la red, características de la aplicación, etc).

El orden de llamada de las opciones no tiene importancia. Si la misma opción se define más de una vez, sólo se tiene en cuenta el valor de la última llamada.

Colecciones

- Presentación de las colecciones
- Conversiones de tipo entre las colecciones y los arrays 4D
- collection.length
- ARRAY TO COLLECTION
- COLLECTION TO ARRAY
- collection.average
- collection.clear
- collection.combine
- collection.concat
- collection.copy
- collection.count
- collection.countValues
- collection.distinct
- collection.equal
- collection.every
- collection.extract
- collection.fill
- collection.filter
- collection.find
- collection.findIndex
- collection.indexOf
- collection.indices
- collection.insert
- collection.join
- collection.lastIndexOf
- collection.map
- collection.max
- collection.min
- collection.orderBy
- collection.orderByMethod
- collection.pop
- collection.push
- collection.query
- collection.reduce
- collection.remove
- collection.resize
- collection.reverse
- collection.shift
- collection.slice
- collection.some
- collection.sort
- collection.sum
- collection.unshift
- New collection
- New shared collection

Presentación de las colecciones

Visión general

Los comandos del tema **Colecciones** crean y trabajan con colecciones.

Las colecciones son listas ordenadas de valores de tipos similares o mixtos (texto, número, objeto, booleano, colección o null). Para manipular las variables del tipo Colección, debe usar la notación del objeto (ver **Uso de la notación objeto**). Para información adicional sobre colecciones 4D, consulte el párrafo **Colección** en la sección **Tipos de datos**.

Para acceder a un elemento de colección, debe pasar el número de elemento entre corchetes:

```
collectionRef[expression]
```

Puede pasar cualquier expresión 4D válida que devuelva un entero positivo en *expression*. Ejemplos:

```
myCollection[5]  //acceso al 6 ° elemento de la colección
myCollection[$var]
```

Nota: tenga en cuenta que los elementos de la colección están numerados desde 0.

Puede asignar un valor a un elemento de colección u obtener un valor de elemento de colección utilizando la notación de objeto:

```
myCol[10]:="My new element"
$myVar:=myCol[0]
```

Si asigna un índice de elemento que supera el último elemento existente de la colección, la colección se redimensionará automáticamente y a todos los elementos intermedios nuevos se les asignará un valor **null**:

```
C_COLLECTION(myCol)
myCol:=New collection("A";"B")
myCol[5]:="Z"
 //myCol[2]=null
 //myCol[3]=null
 //myCol[4]=null
```

Colección estándar o colección compartida

Puede crear dos tipos de colecciones:

- colecciones **estándar** (no compartidas), utilizando el comando **New collection**.
  Estas colecciones admiten una gran cantidad de tipos de datos, incluidas imágenes y punteros. Se pueden editar sin ningún control de acceso específico.
- colecciones **compartidas**, utilizando el comando **New shared collection**.
  Estas colecciones se pueden compartir entre procesos, incluidos los hilos apropiativos. El acceso a estas colecciones está controlado por estructuras **Use...End use**. Para más información, consulte la página **Objetos y colecciones compartidos**.

Métodos colección

Las referencias de colecciones 4D se benefician de métodos especiales llamados *member functions*. Gracias a la notación objeto (ver **Uso de la notación objeto**), estos métodos se pueden aplicar a las referencias de colección utilizando la siguiente sintaxis:

```
{$result:=}myCollection.memberFunction( {params} )
```

Tenga en cuenta que, incluso si no tiene parámetros, una member function must debe llamarse con paréntesis (), de lo contrario se genera un error de sintaxis.

Por ejemplo:

```
$newCol:=$col.copy()  //copia completa de $col a $newCol
$col.push(10;100)  //agrega 10 y 100 a la colección
```

Algunas member functions devuelven la colección original después de la modificación, para que pueda ejecutar las llamadas en una secuencia:

```
$col:=New collection(5;20)
$col2:=$col.push(10;100).sort()  //$col2=[5,10,20,100]
```

Atención: cuando utilice member functions, debe utilizar rutas de propiedad conformes con ECMA Script, es decir, no puede utilizar ".", "[ ]",
ni espacios en los nombres de propiedades. Por ejemplo, de acuerdo con la sección **Identificadores de propiedades de objetos**, los nombres de propiedades tales como *$o["My.special.property"]* son soportados. Sin embargo, no serán utilizables con member functions:

```
$vmin:=$col.min("My.special.property")  //indefinido
$vmin:=$col.min(["My.special.property"])  //error
```

Parámetro rutaProp

Varias member functions aceptan una *rutaProp* como parámetro. Este parámetro puede contener:

- ya sea un nombre de propiedad del objeto, por ejemplo, "apellido"
- o una ruta de propiedad del objeto, es decir, una secuencia jerárquica de sub-propiedades unidas con caracteres de punto, por ejemplo "empleado.hijo.apellido".

En consecuencia, cuando se espera un parámetro *rutaProp*, utilizando nombres de propiedad que contienen uno o más "." **no es soportado** ya que evitará que 4D analice correctamente la ruta.

Nota: para más información, consulte la sección **Identificadores de propiedades de objetos**.

Conversiones de tipo entre las colecciones y los arrays 4D

Al mover valores entre arrays (tipo de datos fijo) y colecciones (sin tipo de datos), 4D aplica conversiones automáticas según los valores y las declaraciones de tipo de los arrays. Esta sección detalla las conversiones de tipo de colecciones a arrays y viceversa.

Conversión de las colecciones a los arrays

Las siguientes conversiones se aplican a los valores tratados por los siguientes comandos:

- **OB GET ARRAY**
- **COLLECTION TO ARRAY**

Tipo de elemento de colección	null	booleano	Infinito	real	cadena	fecha	imagen	objeto	col
ARRAY TEXT	""	"false" o "true"	"Infinity"	número con separador de punto (si es necesario)	Text	Conversión de fecha en texto en función del parámetro de la base Dates en los objetos	"[object Object]"	"[object Object]"	Ele de col sep por
ARRAY LONGINT	0	0 o 1	comportamiento no definido	redondeado según las reglas de redondeo estándar	0 0 if la cadena no comienza por [0-9,+,-,e,.,x], de lo contrario la conversión estándar. Suporta prefijo de notación hexa 0x	0	0	0	0
ARRAY REAL	0	0 o 1	INF	real	same as **ARRAY LONGINT**	0	0	0	0
ARRAY INTEGER	0	0 o 1	0	redondeado de acuerdo con las reglas de redondeo estándar	igual que **ARRAY LONGINT**	0	0	0	0
ARRAY BOOLEAN	False	false or true	true	true if #0	true si string#""	true if date#"00/00/00"	True	True	Tru
ARRAY OBJECT	undefined	undefined	undefined	undefined	undefined	undefined	Objeto imagen	Objeto	Unc
ARRAY PICTURE	0 bytes	0 bytes	0 bytes	0 bytes	0 bytes	0 bytes	Picture	0 bytes	0 b
ARRAY DATE	00/00/00	00/00/00	00/00/00	00/00/00	00/00/00 o una fecha si el formato es cumple con ISO8601	date	00/00/00	00/00/00	00/
ARRAY TIME	00:00:00	00:00:00	comportamiento no definido	número de segundos	número de segundos	00:00:00	00:00:00	00:00:00	00:

Conversión de los arrays a colecciones

Las siguientes conversiones se aplican a los valores tratados por los siguientes comandos:

- **OB SET ARRAY**
- **ARRAY TO COLLECTION**

	ARRAY TEXT	ARRAY LONGINT	ARRAY REAL	ARRAY INTEGER	ARRAY BOOLEAN	ARRAY OBJECT	ARRAY PICTURE	ARRAY DATE	ARRAY TIME
Tipos de elementos de colección	cadena	número	número	número	booleano	objeto o null	imagen	cadena o fecha en función del parámetro de base Datos en los objetos	número de segundos

collection.length

Parámetro	Tipo		Descripción
collection.length	Entero largo		Número de elementos en la colección

Descripción

La propiedad **collection.length** devuelve el número de elementos en la colección.

La propiedad **collection.length** se inicializa cuando se crea la colección. Agregar o quitar elementos actualiza la longitud, si es necesario. Esta propiedad es de solo lectura (no puede usarla para definir el tamaño de la colección).

Ejemplo

```
C_COLLECTION($col)  //$col.length initialized to 0
$col:=New collection("one";"two";"three")  //$col.length updated to 3
$col[4]:="five"  //$col.length updated to 5
$vSize:=$col.remove(0;3).length  //$vSize=2
```

ARRAY TO COLLECTION

ARRAY TO COLLECTION (coleccion ; array {; nombreProp}{; array2 ; nombreProp2 ; ... ; arrayN ; nombrePropN})

Parámetro	Tipo		Descripción
coleccion	Collection	⬅	Colección a recibir los datos del array
array	Array	➡	Array a copiar a la colección; si se pasó nombreProp, array a copiar a los valores de nombreProp en la colección.
nombreProp	Texto	➡	Nombre de la propiedad del objeto cuyo valor llenar con elementos del array

Descripción

El comando **ARRAY TO COLLECTION** copia uno o más *array*(s) en los elementos o los valores de *nombreProp* de la *coleccion*.

Este comando puede funcionar con una *coleccion* que contiene valores o una *coleccion* que contiene objetos, en cuyo caso los parámetros *nombreProp* son obligatorios.

- Si omite el parámetro *nombreProp*, tel comando copia todos los elementos del *array* a la *coleccion*. Si la *coleccion* no estaba vacía, los elementos existentes se reemplazan y se agregan nuevos elementos si el tamaño del *array* es mayor que la longitud de la *coleccion*. Después de que se ejecuta el comando, la longitud de la *coleccion* es idéntica al tamaño del *array*.
- Si pasa uno o más parámetros *nombreProp*, el comando crea o reemplaza objetos como elementos de *coleccion*. Cada objeto se completa con una propiedad cuyo nombre se ofrece en el parámetro *nombreProp*, y cuyo valor es el elemento de array correspondiente. Si la *coleccion* no estaba vacía, los elementos existentes se reemplazan y se agregan nuevos elementos si el tamaño del *array* es mayor que la colección. Después de que se ejecuta el comando, la longitud de la *coleccion* es la misma que el tamaño del array más grande.

Ejemplo 1

Desea copiar un array de texto en una colección:

```
C_COLLECTION($colFruits)
$colFruits:=New collection
ARRAY TEXT($artFruits;4)
$artFruits{1}:="Orange"
$artFruits{2}:="Banana"
$artFruits{3}:="Apple"
$artFruits{4}:="Grape"
ARRAY TO COLLECTION($colFruits;$artFruits)
 //$colFruits[0]="Orange"
 //$colFruits[1]="Banana"
 //...
```

Ejemplo 2

Desea copiar valores de campo en una colección de objetos por medio de arrays:

```
C_COLLECTION($col)
ARRAY TEXT($artCity;0)
ARRAY LONGINT($arLZipCode;0)
SELECTION TO ARRAY([Customer]City;$artCity)
SELECTION TO ARRAY([Customer]Zipcode;$arLZipCode)
ARRAY TO COLLECTION($col;$artCity;"cityName";$arLZipCode;"Zip")
 //$col[0]={"cityName":"Cleveland","Zip":35049}
 //$col[1]={"cityName":"Blountsville","Zip":35031}
 //...
```

Ejemplo 3

Desea copiar una array texto en una colección compartida:

```
ARRAY TEXT($at;1)

APPEND TO ARRAY($at;"Apple")
APPEND TO ARRAY($at;"Orange")
APPEND TO ARRAY($at;"Grape")

C_COLLECTION($sharedCol)
$sharedCol:=New shared collection
```

```
Use($sharedCol)
    ARRAY TO COLLECTION($sharedCol;$at)
End use
```

COLLECTION TO ARRAY

COLLECTION TO ARRAY (coleccion ; array {; nombreProp}{; array2 ; nombreProp2 ; ... ; arrayN ; nombrePropN})

Parámetro	Tipo		Descripción
coleccion	Collection	➡	Colección a copiar en array(s)
array	Array	⬅	Array para recibir los elementos de la colección; si se pasó nombreProp, array para recibir los valores de nombreProp en la colección
nombreProp	Texto	➡	Nombre de la propiedad del objeto cuyos valores copiar en array ("" para todos los elementos)

Descripción

El comando **COLLECTION TO ARRAY** llena una o más *array*(s) con elementos o valores de *nombreProp* de la *coleccion* en *array*(s).

Este comando puede funcionar con una *coleccion* que contiene valores o una *coleccion* que contiene objetos, en cuyo caso los parámetros *nombreProp* son obligatorios.

- Si omite el parámetro *nombreProp*, el comando copia todos los elementos de *coleccion* a *array*. Después de que se ejecuta el comando, el tamaño de *array* es idéntico a la longitud de la *coleccion*.
- Si pasa uno o más parámetros *nombreProp*(s), *coleccion* debe ser una colección de objetos (se ignoran otros elementos). En este caso, cada parámetro *nombreProp* indica el nombre de una propiedad dentro de cada objeto de la colección cuyos valores desea copiar en el *array* correspondiente. Puede pasar todo par *nombreProp* / *array*, combinando tipos de array. Después de que se ejecuta el comando, cada tamaño de *array* es idéntico a la longitud de *coleccion*.

En todos los casos, 4D convierte los elementos o valores de la colección de acuerdo al tipo de *array* (si es necesario). Las reglas de conversión se detallan en la página **Conversiones de tipo entre las colecciones y los arrays 4D**.

Ejemplo 1

Desea copiar una colección de cadenas en un array de texto:

```
C_COLLECTION($fruits)
$fruits:=New collection("Orange";"Banana";"Apple";"Grape")
ARRAY TEXT($artFruits;0)
COLLECTION TO ARRAY($fruits;$artFruits)
 //$artFruits{1}="Orange"
 //$artFruits{2}="Banana"
 //...
```

Ejemplo 2

Desea copiar diferentes valores de propiedad de una colección de objetos en diferentes arrays:

```
C_COLLECTION($col)
$col:=New collection
ARRAY TEXT($city;0)
ARRAY LONGINT($zipCode;0)
$col.push(New object("name";"Cleveland";"zc";35049))
$col.push(New object("name";"Blountsville";"zc";35031))
$col.push(New object("name";"Adger";"zc";35006))
$col.push(New object("name";"Clanton";"zc";35046))
$col.push(New object("name";"Shelby";"zc";35143))

COLLECTION TO ARRAY($col;$city;"name";$zipCode;"zc")
 //$city{1}="Cleveland", $zipCode{1}=35049
 //$city{2}="Blountsville", $zipCode{2}=35031
 //...
```

collection.average( )

collection.average ({rutaProp}) -> Resultado

Parámetro	Tipo		Descripción
rutaProp	Texto	➡	Ruta de la propiedad del objeto que se utilizará para el cálculo
Resultado	Real, Undefined	➲	Media aritmética (promedio) de los valores de colección

Descripción

La función **collection.average( )** devuelve la media aritmética (promedio) de los valores definidos en la instancia de colección.
Solo se tienen en cuenta elementos numéricos para el cálculo (se ignoran otros tipos de elementos).
Si la colección contiene objetos, pase el parámetro *rutaProp* para indicar la propiedad del objeto a tener en cuenta.
collection.average( ) devuelve *Indefinido* si:

- la colección está vacía,
- la colección no contiene elementos numéricos,
- *rutaProp* no se encuentra en la colección.

Ejemplo 1

```
C_COLLECTION($col)
$col:=New collection(10;20;"Monday";True;6)
$vAvg:=$col.average() //12
```

Ejemplo 2

```
C_COLLECTION($col)
$col:=New collection
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Gross";"salary";10500))
$vAvg:=$col.average("salary") //23500
```

collection.clear( )

collection.clear () -> Resultado

Parámetro	Tipo		Descripción
Resultado	Collection		Colección original con todos los elementos eliminados

Descripción

La función **collection.clear( )** elimina todos los elementos de la instancia de colección y devuelve una colección vacía.
Nota: este comando modifica la colección original.

Ejemplo

```
C_COLLECTION($col)
$col:=New collection(1;2;5)
$col.clear()
$vSize:=$col.length  //$vSize=0
```

collection.combine( )

collection.combine (col2 {; index}) -> Resultado

Parámetro	Tipo		Descripción
col2	Collection	➡	Colección a combinar
index	Entero largo	➡	Posición a la que insertar elementos para combinar en la colección (por defecto=longitud+1)
Resultado	Collection	⮌	Colección original que contiene elemento(s) combinado(s)

Descripción

El método **collection.combine( )** inserta elementos *col2* al final o en la posición *index* especificada en la instancia de colección y devuelve la colección editada. A diferencia del método **collection.insert( )**, **collection.combine( )** agrega cada valor de *col2* en la colección original y no como un único elemento de colección.

Nota: este método modifica la colección original.

Por defecto, los elementos *col2* se agregan al final de la colección original. Puede pasar en *index* la posición donde desea insertar los elementos de *col2* en la colección. **Advertencia**: tenga en cuenta que los elementos de la colección están numerados desde 0.

- Si *index* > la longitud de la colección, el índice de inicio real se establecerá para la longitud de la colección.
- Si *index* < 0, se vuelve a calcular como *index:=index+longitud* (se considera como el desplazamiento desde el final de la colección).
- Si el valor calculado es negativo, *index* se establece en 0.

Ejemplo

```
C_COLLECTION($c;$fruits)
$c:=New collection(1;2;3;4;5;6)
$fruits:=New collection("Orange";"Banana";"Apple";"Grape")
$c.combine($fruits;3)  //[1,2,3,"Orange","Banana","Apple","Grape",4,5,6]
```

collection.concat( )

collection.concat (valor {; valor2 ; ... ; valorN}) -> Resultado

Parámetro	Tipo		Descripción
valor	Número, Texto, Objeto, Collection, Fecha, Booleano	➡	Valor(es) a concatenar Si el valor es una colección, todos los elementos de la colección se agregan a la colección original
Resultado	Collection	↩	Colección original con valor(es) añadido(s)

Descripción

El método **collection.concat( )** devuelve una nueva colección con todos los elementos del parámetro *valor* agregado al final de la colección original.

Nota: este comando no modifica la colección original.

Si *valor* es una colección, todos sus elementos se agregan como elementos nuevos al final de la colección original. Si *valor* no es una colección, se agrega como un nuevo elemento.

Ejemplo

```
C_COLLECTION($c)
$c:=New collection(1;2;3;4;5)
$fruits:=New collection("Orange";"Banana";"Apple";"Grape")
$fruits.push(New object("Intruder";"Tomato"))
$c2:=$c.concat($fruits) //[1,2,3,4,5,"Orange","Banana","Apple","Grape",{"Intruder":"Tomato"}]
$c2:=$c.concat(6;7;8) //[1,2,3,4,5,6,7,8]
```

collection.copy( )

collection.copy ({resolvPtrs}) -> Resultado

Parámetro	Tipo		Descripción
resolvPtrs	Entero largo	➡	True = resuelve punteros, False u omitido = no resuelve punteros
Resultado	Collection	⮌	Copia completa de la colección original

Descripción

La función **collection.copy( )** devuelve una copia completa de la instancia de recopilación. *Copia completa* significa que los objetos o colecciones dentro de la colección original están duplicados y no comparten ninguna referencia con la colección devuelta.

Notas:

- Esta función no modifica la colección original.
- Si se aplica a una colección compartida, **copy( )** devuelve una colección regular (no compartida).

Si la colección original contiene valores de tipo de puntero, de manera predeterminada, la copia también contiene los punteros. Sin embargo, puede resolver punteros al momento de la copia: pase la constante ck resolve pointers en el parámetro *resolvPtrs*. En este caso, cada puntero presente como valor en la colección será evaluado al momento de la copia y será utilizado su valor desreferenciado.

Ejemplo

```
C_COLLECTION($col)
C_POINTER($p)
$p:=->$what

$col:=New collection
$col.push(New object("alpha";"Hello";"num";1))
$col.push(New object("beta";"You";"what";$p))

$col2:=$col.copy()
$col2[1].beta:="World!"
ALERT($col[0].alpha+" "+$col2[1].beta)  //muestra "Hello World!"

$what:="You!"
$col3:=$col2.copy(ck resolve pointers)
ALERT($col3[0].alpha+" "+$col3[1].what)  //muestra "Hello You!"
```

collection.count( )

collection.count ({rutaProp}) -> Resultado

Parámetro	Tipo		Descripción
rutaProp	Texto	➡	Ruta de la propiedad del objeto que se utilizará para el cálculo
Resultado	Real	⮌	Número de elementos en la colección

Descripción

El método **collection.count( )** devuelve la cantidad de elementos no nulos en la colección.
Si la colección contiene objetos, puede pasar el parámetro *rutaProp*. En este caso, solo se tienen en cuenta los elementos que contienen *rutaProp*.

Ejemplo

```
C_COLLECTION($col)
C_REAL($count1;$count2)
$col:=New collection(20;30;Null;40)
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Gross";"salary";10500))
$col.push(New object("lastName";"Henry";"salary";12000))
$count1:=$col.count() //$count1=7
$count2:=$col.count("name") //$count2=3
```

collection.countValues( )

collection.countValues (valor {; rutaProp}) -> Resultado

Parámetro	Tipo		Descripción
valor	Texto, Número, Booleano, Fecha, Objeto, Collection	➡	Valor a contar
rutaProp	Texto	➡	Ruta de la propiedad del objeto para los valores a contar
Resultado	Entero largo	⮌	Número de ocurrencias del valor

Descripción

La función **collection.countValues( )** devuelve el número de veces que se encuentra el *valor* en la colección.
Puede pasar en *valor*:

- un valor escalar (texto, número, booleano, fecha),
- un objeto o una referencia de colección.

Para un elemento a encontrar, el tipo de *valor* debe ser equivalente al tipo del elemento; la función usa el operador de igualdad.
El parámetro opcional *rutaProp* le permite contar valores dentro de una colección de objetos: pasar en *rutaProp* la ruta de la propiedad cuyos valores desea contar.
Nota: esta función no modifica la colección original.

Ejemplo 1

```
C_COLLECTION($col)
C_LONGINT($vCount)
$col:=New collection(1;2;5;5;5;3;6;4)
$vCount:=$col.countValues(5) // $vCount=3
```

Ejemplo 2

```
C_COLLECTION($col)
C_LONGINT($vCount)
$col:=New collection
$col.push(New object("name";"Smith";"age";5))
$col.push(New object("name";"Wesson";"age";2))
$col.push(New object("name";"Jones";"age";3))
$col.push(New object("name";"Henry";"age";4))
$col.push(New object("name";"Gross";"age";5))
$vCount:=$col.countValues(5;"age") //$vCount=2
```

Ejemplo 3

```
C_COLLECTION($numbers)
C_COLLECTION($letters)
C_LONGINT($vCount)

$letters:=New collection("a";"b";"c")
$numbers:=New collection(1;2;$letters;3;4;5)

$vCount:=$numbers.countValues($letters) //$vCount=1
```

collection.distinct( )

collection.distinct ({rutaProp}{;}{opcion}) -> Resultado

Parámetro	Tipo		Descripción
rutaProp	Texto	➡	Ruta de atributo cuyos distintos valores desea obtener
opcion	Entero largo	➡	ck diacritical: evaluación diacrítica ("A" # "a" por ejemplo)
Resultado	Collection	⮌	Nueva colección con valores distintos

Descripción

El método **collection.distinct( )** devuelve una colección que contiene solo valores distintos (diferentes) de la colección original.

Nota: este método no modifica la colección original.
La colección devuelta se ordena automáticamente. Los valores **Null** no son devueltos.

Si la colección contiene objetos, puede pasar el parámetro *rutaProp* para indicar la propiedad del objeto cuyos valores distintos desea obtener.

Por defecto, se realiza una evaluación no diacrítica. Si desea que la evaluación diferencie entre mayúsculas y minúsculas o para diferenciar los caracteres acentuados, pase la constante ck diacritical en el parámetro *opcion*.

Ejemplo

```
C_COLLECTION($c;$c2)
$c:=New collection
$c.push("a";"b";"c";"A";"B";"c";"b";"b")
$c.push(New object("size";1))
$c.push(New object("size";3))
$c.push(New object("size";1))
$c2:=$c.distinct() //$c2=["a","b","c",{"size":1},{"size":3},{"size":1}]
$c2:=$c.distinct(ck diacritical) //$c2=["a","A","b","B","c",{"size":1},{"size":3},{"size":1}]
$c2:=$c.distinct("size") //$c2=[1,3]
```

collection.equal( )

collection.equal (coleccion2 {; opcion}) -> Resultado

Parámetro	Tipo		Descripción
coleccion2	Collection	➡	Colección a comparar
opcion	Entero largo	➡	ck diacritical: evaluación diacrítica ("A" # "a" por ejemplo)
Resultado	Booleano	⮌	True si las colecciones son idénticas, de lo contrario false

Descripción

El método **collection.equal( )** compara la colección con *coleccion2* y devuelve **true** si son idénticas (comparación completa).
Por defecto, se realiza una evaluación no diacrítica. Si desea que la evaluación diferencie entre mayúsculas y minúsculas o para diferenciar los caracteres acentuados, pase la constante ck diacritical en el parámetro *opcion*.
Nota: los elementos con valores **Null** no son iguales a los elementos *Undefined*.

Ejemplo

```
C_COLLECTION($c;$c2)
C_BOOLEAN($b)

$c:=New collection(New object("a";1;"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"orange");2;3;4)
$b:=$c.equal($c2)  // false

$c:=New collection(New object("1";"a";"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"orange");2;3)
$b:=$c.equal($c2)  // false

$c:=New collection(New object("a";1;"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"Orange");2;3)
$b:=$c.equal($c2)  // true

$c:=New collection(New object("a";1;"b";"orange");2;3)
$c2:=New collection(New object("a";1;"b";"Orange");2;3)
$b:=$c.equal($c2;ck diacritical)  //false
```

collection.every( )

collection.every ({posicIni ;} nomMet {; param {; param2 ; ... ; paramN}}) -> Resultado

Parámetro	Tipo		Descripción
posicIni	Entero largo	➡	Elemento a partir del cual iniciar la evaluación
nomMet	Texto	➡	Nombre del método a llamar para la prueba
param	Expresión	➡	Parámetro(s) a pasar a nomMet
Resultado	Booleano	↩	True si todos los elemento son evaluados con éxito

Descripción

El método **collection.every( )** devuelve **true** si todos los elementos en la colección pasaron con éxito una prueba implementada en el método *nomMet* pasado.

De forma predeterminada, **collection.every( )** prueba toda la colección. Opcionalmente, puede pasar en *posicIni* el índice del elemento desde el que comienza la prueba.

- Si *posicIni* >= longitud de la colección, se devuelve **false**, lo que significa que la colección no se prueba.
- Si *posicIni* < 0, el fin de la colección se considera como punto de inicio del cálculo de la posición (*posicIni :=posicIni +length*).
- Si *posicIni* = 0, se busca en toda la colección (por defecto).

En *nomMet*, pase el nombre del método a usar para evaluar los elementos de la colección, junto con su(s) parámetro(s) en *param* (opcional). *nomMet* puede realizar cualquier prueba, con o sin los parámetros. Este método recibe un parámetro *Object* en $1 y debe definir *$1.result* como **true** para cada elemento que cumple la prueba.

nomMet recibe los siguientes parámetros:

- en *$1.value*: valor del elemento a evaluar
- en *$2*: *param*
- en *$N*...: param2...paramN

nomMet define los siguientes parámetros:

- *$1.result* (booleano): **true** si la evaluación del valor del elemento es exitosa, de lo contrario, es **false**.
- *$1.stop* (booleano, opcional): **true** para detener la retrollamada del método. El valor devuelto es el último calculado.

En todos los casos, en el momento en que la función **collection.every( )** encuentra el primer elemento de la colección que devuelve **false** en *$1.result*, deja de llamar a *nomMet* y devuelve **false**.

Ejemplo 1

```
C_COLLECTION($c)
$c:=New collection
$c.push(5;3;1;4;6;2)
$b:=$c.every("NumberGreaterThan0")  //devuelve true
$c.push(-1)
$b:=$c.every("NumberGreaterThan0")  //devuelve false
```

Con el siguiente método ***NumberGreaterThan0***:

```
$1.result:=$1.value>0
```

Ejemplo 2

Este ejemplo prueba que todos los elementos de una colección son del tipo real:

```
C_COLLECTION($c)
$c:=New collection
$c.push(5;3;1;4;6;2)
$b:=$c.every("TypeLookUp";ls real)  //$b=true
$c:=$c.push(New object("name";"Cleveland";"zc";35049))
$c:=$c.push(New object("name";"Blountsville";"zc";35031))
$b:=$c.every("TypeLookUp";ls real)  //$b=false
```

Con el siguiente método ***TypeLookUp***:

```
C_OBJECT($1)
C_LONGINT($2)
If(Value type($1.value)=$2)
```

```
    $1.result:=True
End if
```

collection.extract( )

collection.extract (rutaProp {; rutaObj}{; rutaProp2 ; rutaObj2 ; ... ; rutaPropN ; rutaObjN}{; opcion}) -> Resultado

Parámetro	Tipo		Descripción
rutaProp	Texto	➡	Ruta de la propiedad del objeto cuyos valores deben extraerse a la nueva colección
rutaObj	Texto	➡	Ruta de la propiedad objetivo o nombre de propiedad
opcion	Entero largo	➡	ck keep null: incluye propiedades nulas en la colección devuelta (ignorada por defecto). El parámetro ignorado si se pasó rutaObj.
Resultado	Collection	↩	Nueva colección que contiene valores extraídos

Descripción

La función **collection.extract( )** crea y devuelve una nueva colección que contiene los valores *rutaProp* extraídos de la colección original de objetos.

Nota: este comando no modifica la colección original.

El contenido de la colección devuelta depende del parámetro *rutaObj*:

- Si se omite el parámetro *rutaObj*, **collection.extract( )** llena la nueva colección con los valores de *rutaProp* de la colección original.
  De forma predeterminada, los elementos para los que *rutaProp* es *null* o indefinido se ignoran en la colección resultante.
  Puede pasar la constante ck keep null en el parámetro *opcion* para incluir estos valores como elementos **null** en la colección devuelta.
- Si se pasan uno o más parámetros *rutaObj*, **collection.extract( )** llena la nueva colección con las propiedades *rutaProp* y cada elemento de la nueva colección es un objeto con las propiedades *rutaObj* llenas con las propiedades *rutaProp*. Los valores nulos se mantienen (el parámetro *opcion* se ignora con esta sintaxis).

Ejemplo 1

```
C_COLLECTION($c)
$c:=New collection
$c.push(New object("name";"Cleveland"))
$c.push(New object("zip";5321))
$c.push(New object("name";"Blountsville"))
$c.push(42)
$c2:=$c.extract("name")  // $c2=[Cleveland,Blountsville]
$c2:=$c.extract("name";ck keep null)  //$c2=[Cleveland,null,Blountsville,null]
```

Ejemplo 2

```
C_COLLECTION($c)
$c:=New collection
$c.push(New object("zc";35060))
$c.push(New object("name";Null;"zc";35049))
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$c2:=$c.extract("name";"City")  //$c2=[{City:null},{City:Cleveland},{City:Blountsville},{City:Adger},{City:Clanton},{City:Clanton}]
$c2:=$c.extract("name";"City";"zc";"Zip")  //$c2=[{Zip:35060},{City:null,Zip:35049},{City:Cleveland,Zip:35049},
{City:Blountsville,Zip:35031},{City:Adger,Zip:35006},{City:Clanton,Zip:35046},{City:Clanton,Zip:35045}]
```

collection.fill( )

collection.fill (valor {; posicIni {; fin}}) -> Resultado

Parámetro	Tipo		Descripción
valor	Número, Texto, Collection, Objeto, Fecha, Booleano	➡	Valor de llenado
posicIni	Entero largo	➡	Elemento de inicio (incluído)
fin	Entero largo	➡	Elemento final (no incluido)
Resultado	Collection	↩	Colección original con valores completados

Descripción

El método **collection.fill( )** llena los elementos de la colección con el *valor* especificado, opcionalmente desde *posicIni a fin* y devuelve la colección resultante.

Nota: este método modifica la colección original.

- Si el parámetro *posicIni* se omite, *valor* se aplica a todos los elementos de colección (*posicIni* =0). Si se omite el parámetro *fin*, el valor se establece en el último elemento de la colección (*fin* =length).
- Si *posicIni* < 0, se recalcula como *posicIni :=posicIni +length* (se considera como el desplazamiento desde el final de la colección). Si el valor calculado es negativo, *posicIni* toma el valor 0.
- Si *fin* < 0 , se recalcula como *fin* :=*fin* +*length*.
- Si *fin d* < *posicIni* (valores pasados o calculados), el método no hace nada.

Ejemplo

```
C_COLLECTION($c)
$c:=New collection(1;2;3;"Lemon";Null;"";4;5)
$c.fill("2") // $c:=[2,2,2,2,2,2,2,2]
$c.fill("Hello";5) // $c=[2,2,2,2,2,Hello,Hello,Hello]
$c.fill(0;1;5) // $c=[2,0,0,0,0,Hello,Hello,Hello]
$c.fill("world";1;-5) //-5+8=3 -> $c=[2,"world","world",0,0,Hello,Hello,Hello]
```

collection.filter( )

collection.filter (nomMet {; param}{; param2 ; ... ; paramN}) -> Resultado

Parámetro	Tipo		Descripción
nomMet	Texto	➡	Nombre de la función a llamar para filtrar la colección
param	Expresión	➡	Parámetro(s) a pasar a nomMet
Resultado	Collection	⮌	Nueva colección que contiene elementos filtrados (copia superficial)

Descripción

El método **collection.filter( )** devuelve una nueva colección que contiene todos los elementos de la colección original para los cuales el resultado del método *nomMet* es **true**. Este método devuelve una *copia superficial*, lo que significa que los objetos o colecciones en ambas colecciones comparten la misma referencia. Si la colección original es una colección compartida, la colección devuelta también es una colección compartida.

Nota: este método no modifica la colección original.

En *nomMet*, pase el nombre del método a utilizar para evaluar los elementos de la colección, junto con su(s) parámetro(s) en *param* (opcional). *nomMet* puede realizar cualquier prueba, con o sin el(los) parámetro(s), y debe devolver **true** en *$1.result* para cada elemento que cumpla la condición y por lo tanto, para avanzar a la nueva colección.

nomMet recibe los siguientes parámetros:

- en *$1.value*: valor del elemento a filtrar
- en *$2*: *param*
- en *$N...*: param2...paramN

nomMet define los siguientes parámetros:

- *$1.result* (booleano): **true** si el valor del elemento coincide con la condición del filtro y debe mantenerse.
- *$1.stop* (booleano, opcional): **true** para detener la retrollamada del método. El valor devuelto es el último calculado.

Ejemplo 1

Desea obtener la colección de elementos de texto cuya longitud es menor que 6:

```
C_COLLECTION($col)
C_COLLECTION($colNew)
$col:=New collection("hello";"world";"red horse";66;"tim";"san jose";"miami")
$colNew:=$col.filter("LengthLessThan";6)
 //$colNew=["hello","world","tim","miami"]
```

El código para el método ***LengthLessThan*** es:

```
C_OBJECT($1)
C_LONGINT($2)
If(Value type($1.value)=Is text)
    $1.result:=(Length($1.value))<$2
End if
```

Ejemplo 2

Desea filtrar los elementos de acuerdo con su tipo de valor:

```
C_COLLECTION($c)
$c:=New collection(5;3;1;4;6;2)
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c2:=$c.filter("TypeLookUp";Is real)  // $c2=[5,3,1,4,6,2]
$c3:=$c.filter("TypeLookUp";Is object)
 // $c3=[{name:Cleveland,zc:35049},{name:Blountsville,zc:35031}]
```

El código para ***TypeLookUp*** es:

```
C_OBJECT($1)
C_LONGINT($2)
If(OB Get type($1;"value")=$2)
    $1.result:=True
End if
```

collection.find( )

collection.find ({posicIni ;} nomMet {; param {; param2 ; ... ; paramN}}) -> Resultado

Parámetro	Tipo		Descripción
posicIni	Entero largo	➡	Número del elemento de inicio de la búsqueda
nomMet	Texto	➡	Nombre de la función a llamar para la búsqueda.
param	Expresión	➡	Parámetro(s) a pasar a nomMet
Resultado		↩	Primer valor encontrado, o -1 si no se encuentra

Descripción

El método **collection.find( )** devuelve el primer valor en la colección para el cual *nomMet*, aplicado en cada elemento, devuelve **true**.

Nota: este método no modifica la colección original.

De forma predetermina, **collection.find( )** busca en toda la colección. Opcionalmente, puede pasar en *posicIni* el índice del elemento desde el cual comenzar la búsqueda.

- Si *posicIni* >= longitud de la colección, se devuelve -1, y la búsqueda no se efectúa.
- Si *posicIni* < 0, se considera como el punto de inicio del cálculo de la posición (*posicIni* :=*posicIni* +length).
  Nota: incluso si *posicIni* es negativo, la colección se sigue buscando de izquierda a derecha.
- Si *posicIni* = 0, se busca en toda la colección (por defecto).

En *nomMet*, pase el nombre del método a usar para evaluar los elementos de la colección, junto con su(s) parámetro(s) en *param* (opcional). *nomMet* puede realizar cualquier prueba, con o sin el(los) parámetro(s). Este método recibe un parámetro *Object* en $1 y debe definir *$1.result* como **true** para el primer elemento que cumpla la condición.

nomMet recibe los siguientes parámetros:

- en *$1.value*: valor del elemento a evaluar
- en *$2*: *param*
- en *$N*...: param2...paramN

nomMet define los siguientes parámetros:

- *$1.result* (booleano): **true** si el valor del elemento coincide con la condición de búsqueda.
- *$1.stop* (booleano, opcional): **true** para detener la retrollamada del método. El valor devuelto es el último calculado.

Ejemplo 1

Usted desea obtener el primer elemento con una longitud inferior a 5:

```
C_COLLECTION($col)
$col:=New collection("hello";"world";4;"red horse";"tim";"san jose")
$value:=$col.find("LengthLessThan";5)  //$value="tim"
```

El código para el método ***LengthLessThan*** es:

```
C_OBJECT($1)
C_LONGINT($2)
If(Value type($1.value)=Is text)
   $1.result:=(Length($1.value))<$2
End if
```

Ejemplo 2

Usted desea encontrar el nombre de una ciudad dentro de una colección:

```
C_COLLECTION($c)
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$c2:=$c.find("FindCity";"Clanton")  //$c2={name:Clanton,zc:35046}
```

El código para el método es ***FindCity***:

```
C_OBJECT($1)
C_TEXT($2)
```

```
$1.result:=$1.value.name=$2
```

collection.findIndex( )

collection.findIndex ({posicIni ;} nomMetodo {; param {; param2 ; ... ; paramN}}) -> Resultado

Parámetro	Tipo		Descripción
posicIni	Entero largo	➡	Índice para comenzar la búsqueda
nomMetodo	Texto	➡	Nombre del método a llamar para la búsqueda
param	Expresión	➡	Parámetro(s) a pasar a nomMet
Resultado	Entero largo	⮌	Índice del primer valor encontrado, o indefinido si no se encuentra

Descripción

El método **collection.findIndex( )** devuelve el índice, en la colección, del primer valor para el cual *nomMet*, aplicado en cada elemento, devuelve **true**. El método devuelve -1 si ningún elemento de la colección se evaluó como true.

Nota: este método no modifica la colección original.

De forma predeterminada, **collection.findIndex( )** busca en toda la colección. Opcionalmente, puede pasar en *posicIni* el índice del elemento desde el cual comenzar la búsqueda.

- Si *posicIni* >= longitud de la colección (**collection.length**), el método devuelve -1 (no se realiza la búsqueda).
- Si *posicIni* < 0, el fin de la colección se considera como el punto de inicio del cálculo de la posición (*posicIni* :=*posicIni* +length).
  Nota: incluso si *posicIni* es negativo, la colección se sigue buscando de izquierda a derecha.
- Si *posicIni* = 0, se busca en toda la colección (por defecto).

En *nomMet*, pase el nombre del método a usar para evaluar los elementos de la colección, junto con su(s) parámetro(s) en *param* (opcional). *nomMet* puede realizar cualquier prueba, usando o no los parámetros. Este método recibe un parámetro *Object* en $1 y debe establecer *$1.result* como **true** para el primer elemento que cumpla la condición.

nomMet recibe los siguientes parámetros:

- en *$1.value*: valor del elemento a evaluar
- en *$2*: *param*
- en *$N*...: param2...paramN

nomMet define los siguientes parámetros:

- *$1.result* (booleano): **true** si el valor del elemento coincide con la condición de búsqueda.
- *$1.stop* (booleano, opcional): **true** para detener la retrollamada del método. El valor devuelto es el último calculado.

Ejemplo

Usted desea encontrar la posición del primer nombre de la ciudad dentro de una colección:

```
C_COLLECTION($c)
C_LONGINT($val2;$val3)
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$val2:=$c.findIndex("FindCity";"Clanton") // $val2=3
$val3:=$c.findIndex($val2+1;"FindCity";"Clanton") //$val3=4
```

El código para el método es ***FindCity***:

```
C_OBJECT($1)
C_TEXT($2)
$1.result:=$1.value.name=$2
```

collection.indexOf( )

collection.indexOf (aBuscar {; posicIni}) -> Resultado

Parámetro	Tipo		Descripción
aBuscar	Expresión	➡	Elemento a buscar en la colección
posicIni	Entero largo	➡	Número del elemento a partir del cual iniciar la búsqueda
Resultado	Entero largo	⮌	Número de la primera ocurrencia de aBuscar en la colección, -1 si no se encuentra

Descripción

El método **collection.indexOf( )** busca la expresión *aBuscar* entre los elementos de la colección y devuelve el número del elemento de la primera ocurrencia encontrada, o -1 si no se encuentra.

Nota: este método no modifica la colección original.

En *aBuscar*, pase la expresión a buscar en la colección. Puedes pasar:

- un valor escalar (texto, número, booleano, fecha),
- el valor null,
- un objeto o una referencia de colección.

aBuscar debe coincidir exactamente con el elemento a buscar (se aplican las mismas reglas que para el operador de igualdad, ver **Operadores de comparación**).

Opcionalmente, puede pasar el número del elemento desde el cual iniciar la búsqueda en *posicIni*.

- Si *posicIni* >= longitud de la colección, se devuelve -1, lo que significa que no se busca en la colección.
- Si *posicIni* < 0, el fin de la colección se considera como punto de inicio del cálculo de la posición (*posicIni* :=*posicIni* +length).
  Nota: incluso si *posicIni* es negativo, la colección se sigue buscando de izquierda a derecha.
- Si *posicIni* = 0, se busca en toda la colección (por defecto).

Ejemplo

```
C_COLLECTION($col)
$col:=New collection(1;2;"Henry";5;3;"Albert";6;4;"Alan";5)
$i:=$col.indexOf(3) //$i=4
$i:=$col.indexOf(5;5) //$i=9
$i:=$col.indexOf("al@") //$i=5
$i:=$col.indexOf("Hello") //$i=-1
```

collection.indices( )

collection.indices (cadenaBusq {; value}{; value2 ; ... ; valueN}) -> Resultado

Parámetro	Tipo		Descripción
cadenaBusq	Texto	➡	Cadena que contiene los criterios de búsqueda
value	Mixed	➡	Value(s) to compare when using placeholder(s)
Resultado	Collection	⮌	Número de elementos de la colección que responden a los criterios de búsqueda

Descripción

El método **collection.indices( )** funciona exactamente igual que el método **collection.query( )** pero devuelve las **posiciones**, en la colección de origen, de los elementos que coinciden con los criterios de búsqueda de *cadenaBusq* y no elementos en sí mismos.

Nota: este método no modifica la colección original.

El parámetro *cadenaBusq* utiliza la siguiente sintaxis:

```
propertyPath comparator value {logicalOperator propertyPath comparator value}
```

Para una descripción de los parámetros *cadenaBusq* y *valor*, consulte el método **dataClass.query( )**.

Ejemplo

```
C_COLLECTION($c)
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$icol:=$c.indices("name = :1";"Cleveland")  // $icol=[0]
$icol:=$c.indices("zc > 35040")  // $icol=[0,3,4]
```

collection.insert( )

collection.insert (indice ; elemento) -> coleccion

Parámetro	Tipo		Descripción
indice	Entero largo	➡	Donde insertar el elemento
elemento	Expresión	➡	Elemento a insertar en la colección
coleccion	Collection	⮌	Colección original que contiene el elemento insertado

Descripción

La función **collection.insert( )** inserta el *elemento* en la posición de *indice* especificada en la instancia de colección y devuelve la colección editada.

Nota: este comando modifica la colección original.

En *indice*, pase la posición donde desea insertar el elemento en la colección.

Atención: tenga en cuenta que los elementos de la colección están numerados desde 0.

- Si *indice* > la longitud de la colección, el índice de inicio real se establecerá en la longitud de la colección.
- Si *indice* <0, se vuelve a calcular como *indice:=indice+longitud* (se considera como el desplazamiento desde el final de la colección).
- Si el valor calculado es negativo, *indice* se establece en 0.

Se puede insertar cualquier tipo de *elemento* aceptado por una colección, incluso otra colección.

Ejemplo

```
C_COLLECTION($col)
$col:=New collection("a";"b";"c";"d") //$col=["a","b","c","d"]
$col.insert(2;"X") //$col=["a","b","X","c","d"]
$col.insert(-2;"Y") //$col=["a","b","X","Y","c","d"]
$col.insert(-10;"Hi") //$col=["Hi","a","b","X","Y","c","d"]
```

collection.join( )

collection.join (delimitador {; opción}) -> Resultado

Parámetro	Tipo		Descripción
delimitador	Texto	➡	Separador a utilizar entre cada elemento
opción	Entero largo	➡	ck ignora null o vacío: ignora cadenas null y vacías en el resultado
Resultado	Texto	⮌	Cadena que contiene todos los elementos de la colección, separados por delimitador

Descripción

El método **collection.join( )** convierte todos los elementos de la colección en cadenas y los concatena utilizando la cadena *delimitador* especificada como separador. El método devuelve la cadena resultante.

Nota: este método no modifica la colección original.

Por defecto, los elementos null o vacíos de la colección se devuelven en la cadena resultante. Pase la constante ck ignore null or empty en el parámetro *opcion* si desea eliminarlos de la cadena resultante.

Ejemplo

```
C_COLLECTION($c)
C_TEXT($t1;$t2)
$c:=New collection(1;2;3;"Paris";Null;"";4;5)
$t1:=$c.join("|")  //1|2|3|Paris|null|||4|5
$t2:=$c.join("|";ck ignore null or empty)  //1|2|3|Paris|4|5
```

collection.lastIndexOf( )

collection.lastIndexOf (aBuscar {; posicIni}) -> Resultado

Parámetro	Tipo		Descripción
aBuscar	Expresión	➡	Elemento a buscar en la colección
posicIni	Entero largo	➡	Número del elemento a partir del cual iniciar la búsqueda
Resultado	Entero largo	⮌	Número de la última ocurrencia de aBuscar en la colección, -1 si no se encuentra

Descripción

El método **collection.lastIndexOf( )** busca la expresión *aBuscar* entre los elementos de la colección y devuelve el índice de la última ocurrencia, o -1 si no se encontró.

Nota: este método no modifica la colección original.

En *aBuscar*, pase la expresión a buscar en la colección. Puedes pasar:

- un valor escalar (texto, número, booleano, fecha),
- el valor nulo,
- una referencia de objeto o de colección.

aBuscar debe coincidir exactamente con el elemento a buscar (se aplican las mismas reglas que para el operador de igualdad, ver **Operadores de comparación**).

Opcionalmente, puede efectuar una búsqueda en sentido inverso pasando el número del elemento en el cual iniciar la búsqueda en *posicIni*.

- Si *posicIni* >= la longitud de la colección menos uno (coll.length-1), se busca en toda la colección (por defecto).
- Si *posicIni* < 0, se recalcula como *posicIni* :=*posicIni* +length (el fin de la colección se considera como punto de inicio del calculo de la posición). Si el valor calculado es negativo, se devuelve -1 (la colección no se evalúa).
  Nota: incluso si *posicIni* es negativo, la colección se evalúa de derecha a izquierda.
- Si *posicIni* = 0, -1 se devuelve -1, lo que significa que no se busca la colección.

Ejemplo

```
C_COLLECTION($col)
$col:=Split string("a,b,c,d,e,f,g,h,i,j,e,k,e";",") //$col.length=13
$pos1:=$col.lastIndexOf("e") //devuelve 12
$pos2:=$col.lastIndexOf("e";6) //devuelve 4
$pos3:=$col.lastIndexOf("e";15) //devuelve 12
$pos4:=$col.lastIndexOf("e";-2) //devuelve 10
$pos5:=$col.lastIndexOf("x") //devuelve -1
```

collection.map( )

collection.map (nomMet ; param {; param2 ; ... ; paramN}) -> Resultado

Parámetro	Tipo		Descripción
nomMet	Texto	➡	Nombre del método utilizado para transformar los elementos de la colección
param	Expresión	➡	Parámetros para el método
Resultado	Collection	⮌	Colección de valores transformados

Descripción

El método **collection.map( )** crea una nueva colección basada en el resultado de la llamada del método *nomMet* en cada elemento de la colección original. Opcionalmente, puede pasar parámetros a *nomMet* usando los parámetros *param*. **collection.map( )** siempre devuelve una colección del mismo tamaño que la colección original.

nomMet recibe los siguientes parámetros:

- en *$1.value* (todo tipo): valor del elemento a ser mapeado
- en *$2* (todo tipo): *param*
- en *$N*... (todo tipo): *param2*...*paramN*

nomMet define los siguientes parámetros:

- *$1.result* (todo tipo): nuevo valor transformado para agregar a la colección resultante
- *$1.stop* (booleano): **true** para detener la retollamada del método. El valor devuelto es el último calculado.

Ejemplo

```
C_COLLECTION($c;$c2)
$c:=New collection(1;4;9;10;20)
$c2:=$c.map("Percentage";$c.sum())
 //$c2=[2.27,9.09,20.45,22.73,45.45]
```

Este es el método ***Percentage***:

```
C_OBJECT($1)
C_REAL($2)
$1.result:=Round(($1.value/$2)*100;2)
```

collection.max( )

collection.max ({rutaProp}) -> Resultado

Parámetro	Tipo		Descripción
rutaProp	Texto	➡	Ruta de la propiedad del objeto que se utilizará para la evaluación
Resultado	Booleano, Collection, Fecha, Número, Objeto, Texto	↩	Máximo valor en la colección

Descripción

El método **collection.max( )** devuelve el elemento con el valor más alto en la colección (el último elemento de la colección, ya que se ordenaría en orden ascendente con el método **collection.sort( )**.

Nota: esta función no modifica la colección original.

Si la colección contiene diferentes tipos de valores, la función **max( )** devolverá el valor máximo dentro del último tipo de elemento en el orden de la lista de tipos (ver la descripción **collection.sort( )**).

Si la colección contiene objetos, pase el parámetro *rutaProp* para indicar la propiedad del objeto cuyo valor máximo desea obtener.

Si la colección está vacía, **collection.max( )** devuelve *Undefined*.

Ejemplo

```
C_COLLECTION($col)
$col:=New collection(200;150;55)
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Alabama";"salary";10500))
$max:=$col.max() //{name:Alabama,salary:10500}
$maxSal:=$col.max("salary") //50000
$maxName:=$col.max("name") //"Wesson"
```

collection.min( )

collection.min ({rutaProp}) -> Resultado

Parámetro	Tipo		Descripción
rutaProp	Texto	➡	Ruta de la propiedad del objeto que se utilizará para la evaluación
Resultado	Booleano, Collection, Fecha, Número, Objeto, Texto	⮌	Valor mínimo en la colección

Descripción

El método **collection.min( )** devuelve el elemento con el valor más pequeño en la colección (el primer elemento de la colección, ya que se ordenaría en orden ascendente utilizando el método **collection.sort( )**).

Nota: este método no modifica la colección original.

Si la colección contiene diferentes tipos de valores, el método **collection.min( )** devolverá el valor mínimo dentro del primer tipo de elemento en el orden de la lista de tipos (ver descripción **collection.sort( )**).

Si la colección contiene objetos, pase el parámetro *rutaProp* para indicar la propiedad del objeto cuyo valor mínimo desea obtener.

Si la colección está vacía, **collection.min( )** devuelve *Indefinido*.

Ejemplo

```
C_COLLECTION($col)
$col:=New collection(200;150;55)
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Alabama";"salary";10500))
$min:=$col.min() //55
$minSal:=$col.min("salary") //10000
$minName:=$col.min("name") //"Alabama"
```

collection.orderBy( )

collection.orderBy ({criterio}) -> Resultado

Parámetro	Tipo		Descripción
criterio	Texto, Collection, Entero largo	➡	Texto: ruta(s) de propiedad(es) para ordenar la colección Colección: colección de objetos de criterio Entero largo: ck ascending o ck descending (valores escalares)
Resultado	Collection	⮌	Copia ordenada de la colección (copia superficial)

Descripción

El método **collection.orderBy( )** devuelve una nueva colección que contiene todos los elementos de la colección en el orden especificado por *criterio*.

Este método devuelve una *copia superficial*, lo que significa que los objetos o colecciones en ambas colecciones comparten la misma referencia. Si la colección original es una colección compartida, la colección devuelta también es una colección compartida.

Nota: este método no modifica la colección original.

Si omite el parámetro *criterio*, el método ordena los valores escalares en la colección en orden ascendente (otros tipos de elementos como objetos o colecciones se devuelven desordenados). Puede modificar este orden automático pasando las constantes ck ascending o ck descending en el parámetro *criterio* (ver abajo).

También puede pasar un parámetro *criterio* para definir cómo deben ordenarse los elementos de la colección. Tres sintaxis son compatibles para este parámetro:

- *criterio* es del **tipo texto** (fórmula): "rutaPropiedad1 {desc o asc}, rutaPropiedad2 {desc o asc}, ..." (orden por defecto: *asc*)
  En este caso, *criterio* contiene una fórmula formada de 1 a x rutas de propiedad y (opcionalmente) órdenes de clasificación, separados por comas. El orden en que se pasan las propiedades determina la prioridad de clasificación de los elementos de la colección.
  Por defecto, las propiedades se ordenan en orden ascendente. Puede establecer el orden de clasificación de una propiedad en la cadena de criterios, separada de la ruta de la propiedad por un espacio único: pase "asc" para ordenar en orden ascendente o "desc" en orden descendente.

- *criterio* es de **tipo de colección**: en este caso, cada elemento de la colección contiene un objeto estructurado de la siguiente manera:
  {"rutaPropiedad": cadena,
  "descendiente": booleano}
  Por defecto, las propiedades se ordenan en orden ascendente ("descendente" es falso).
  Puede agregar tantos objetos en la colección *criterio* como sea necesario.

- *criterio* es de **tipo entero largo**: en este caso, pase una de las siguientes constantes del tema **Objetos y colecciones**:

Constante	Tipo	Valor	Comentario
ck ascending	Entero largo	0	Los elementos se ordenan de forma ascendente (predeterminado)
ck descending	Entero largo	1	Los elementos se ordenan en orden descendente

  Esta sintaxis solo ordena valores escalares en la colección (otros tipos de elementos como objetos o colecciones se devuelven desordenados).

Si la colección contiene elementos de diferentes tipos, primero se agrupan por tipo y se ordenan después. Los tipos se devuelven en el siguiente orden:

1. nulo
2. booleanos
3. cadenas
4. números
5. objetos
6. colecciones
7. fechas

Ejemplo 1

Ordenar una colección de números en orden ascendente y descendente:

```
C_COLLECTION($c;$c2;$3)
$c:=New collection
For($vCounter;1;10)
   $c.push(Random)
End for
$c2:=$c.orderBy(ck ascending)
$c3:=$c.orderBy(ck descending)
```

Ejemplo 2

Ordenar una colección de objetos basada en una fórmula de texto con nombres de propiedad:

```
C_COLLECTION($c)
$c:=New collection
For($vCounter;1;10)
    $c.push(New object("id";$vCounter;"value";Random))
End for
$c2:=$c.orderBy("value desc")
$c2:=$c.orderBy("value desc, id")
$c2:=$c.orderBy("value desc, id asc")
```

Ordenar una colección de objetos con una ruta de propiedad:

```
C_COLLECTION($c)
$c:=New collection
$c.push(New object("name";"Cleveland";"phones";New object("p1";"01";"p2";"02")))
$c.push(New object("name";"Blountsville";"phones";New object("p1";"00";"p2";"03")))
$c2:=$c.orderBy("phones.p1 asc")
```

Ejemplo 3

Ordenar una colección de objetos utilizando una colección de objetos criterio:

```
C_COLLECTION($crit;$c)
$crit:=New collection
$c:=New collection
For($vCounter;1;10)
    $c.push(New object("id";$vCounter;"value";Random))
End for
$crit.push(New object("propertyPath";"value";"descending";True))
$crit.push(New object("propertyPath";"id";"descending";False))
$c2:=$c.orderBy($crit)
```

Ordenar con una ruta de propiedad:

```
C_COLLECTION($crit;$c)
$c:=New collection
$c.push(New object("name";"Cleveland";"phones";New object("p1";"01";"p2";"02")))
$c.push(New object("name";"Blountsville";"phones";New object("p1";"00";"p2";"03")))
$crit:=New collection(New object("propertyPath";"phones.p2";"descending";True))
$c2:=$c.orderBy($crit)
```

collection.orderByMethod( )

collection.orderByMethod (nomMet {; extraParam}{; extraParam2 ; ... ; extraParamN}) -> Resultado

Parámetro	Tipo		Descripción
nomMet	Texto	➡	Nombre del método utilizado para ordenar la colección
extraParam	Expresión	➡	Parámetro(s) para el método
Resultado	Collection	⮌	Copia ordenada de la colección (copia superficial)

Descripción

El método **collection.orderByMethod( )** devuelve una nueva colección que contiene todos los elementos de la colección en el orden definido por el método *nomMet*.

Este método devuelve una copia *superficial*, lo que significa que los objetos o colecciones en ambas colecciones comparten la misma referencia. Si la colección original es una colección compartida, la colección devuelta también es una colección compartida.

Nota: este método no modifica la colección original.

En *nomMet*, pase un método de comparación que compare dos valores y devuelva **true** en *$1.result* si el primer valor es menor que el segundo valor. Puede ofrecer parámetros adicionales a *nomMet* si es necesario.

- *nomMet* recibirá los siguientes parámetros:
  - $1 (objeto), donde:
    - *$1.value* (todo tipo): valor del primer elemento a comparar
    - *$1.value2* (todo tipo): valor del segundo elemento a comparar
  - $2...$N (todo tipo): parámetros adicionales
- *nomMet* define el siguiente parámetro:
  - *$1.result* (booleano): **true** si *$1.value* < *$1.value2*, de lo contrario false

Ejemplo 1

Usted desea ordenar una colección de cadenas en orden numérico en lugar de orden alfabético:

```
C_COLLECTION($c)
$c:=New collection
$c.push("33";"4";"1111";"222")
$c2:=$c.orderBy() //$c2=["1111","222","33","4"], alphabetical order
$c3:=$c.orderByMethod("NumAscending") // $c3=["4","33","222","1111"]
```

Aquí está el código para ***NumAscending***:

```
$1.result:=Num($1.value)<Num($1.value2)
```

Ejemplo 2

Usted desea ordenar una colección de cadenas por su longitud:

```
C_COLLECTION($fruits)
$fruits:=New collection("Orange";"Apple";"Grape";"pear";"Banana";"fig";"Blackberry";"Passion fruit")
$c2:=$fruits.orderByMethod("WordLength")
 //$c2=[Passion fruit,Blackberry,Orange,Banana,Apple,Grape,pear,fig]
```

Aquí está el código para ***WordLength***:

```
$1.result:=Length(String($1.value))>Length(String($1.value2))
```

collection.pop( )

collection.pop () -> Resultado

Parámetro	Tipo	Descripción
Resultado	Expresión	Último elemento de la colección

Descripción

La función **collection.pop( )** elimina el último elemento de la colección y lo devuelve como resultado de la función.
Nota: este comando modifica la colección original.
Cuando se aplica a una colección vacía, **collection.pop( )** devuelve ***indefinido***.

Ejemplo

collection.pop( ), utilizado junto con **collection.push( )**, se puede usar para implementar una funcionalidad de pila de primero en entrar, último en salir:

```
C_COLLECTION($stack)
$stack:=New collection  //$stack=[]
$stack.push(1;2)  //$stack=[1,2]
$stack.pop()  //$stack=[1]  Returns 2
$stack.push(New collection(4;5))  //$stack=[[1,[4,5]]
$stack.pop()  //$stack=[1]  Returns [4,5]
$stack.pop()  //$stack=[]  Returns 1
```

collection.push( )

collection.push (elemento {; elemento2 ; ... ; elementoN}) -> Resultado

Parámetro	Tipo		Descripción
elemento	Expresión	➡	Elemento(s) para añadir a la colección
Resultado	Collection	⮌	Colección original que contiene los elementos agregados

Descripción

La función **collection.push( )** agrega uno o más *elemento*(s) al final de la instancia de colección y devuelve la colección editada.
Nota: este comando modifica la colección original.

Ejemplo 1

```
C_COLLECTION($col)
$col:=New collection(1,2) //$col=[1,2]
$col.push(3) //$col=[1,2,3]
$col.push(6;New object("firstname";"John";"lastname";"Smith"))
 //$col=[1,2,3,6,{firstname:John,lastname:Smith}
```

Ejemplo 2

Desea ordenar la colección resultante:

```
C_COLLECTION($col;$sortedCol)
$col:=New collection(5;3;9) //$col=[5,3,9]
$sortedCol:=$col.push(7;50).sort()
 //$col=[5,3,9,7,50]
 //$sortedCol=[3,5,7,9,50]
```

collection.query( )

collection.query (cadenaBusq {; valor}{; valor2 ; ... ; valorN}) -> Resultado

Parámetro	Tipo		Descripción
cadenaBusq	Texto	➡	Cadena que contiene los criterios de búsqueda
valor	Mixed	➡	Valor(es) a comparar al utilizar separador(es)
Resultado	Collection	⮌	Elementos que responden a los criterios de búsqueda

Descripción

El método **collection.query( )** devuelve todos los elementos de una colección que coinciden con las condiciones de búsqueda definidas por *cadenaBusq*. Si la colección original es una colección compartida, la colección devuelta también es una colección compartida.

Nota: este método no modifica la colección original.

El parámetro *cadenaBusq* utiliza la siguiente sintaxis:

```
propertyPath comparator value {logicalOperator propertyPath comparator value}
```

Para obtener información detallada sobre cómo crear una consulta utilizando los parámetros *cadenaBusq* y *valor*, consulte la descripción del método **dataClass.query( )**.

Ejemplo 1

```
C_COLLECTION($c)
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.push(New object("name";"Blountsville";"zc";35031))
$c.push(New object("name";"Adger";"zc";35006))
$c.push(New object("name";"Clanton";"zc";35046))
$c.push(New object("name";"Clanton";"zc";35045))
$c2:=$c.query("name = :1";"Cleveland") //$c2=[{name:Cleveland,zc:35049}]
$c3:=$c.query("zc > 35040") //$c3=[{name:Cleveland,zc:35049},{name:Clanton,zc:35046},{name:Clanton,zc:35045}]
```

Ejemplo 2

```
C_COLLECTION($c)
$c:=New collection
$c.push(New object("name";"Smith";"dateHired";!22-05-2002!;"age";45))
$c.push(New object("name";"Wesson";"dateHired";!30-11-2017!))
$c.push(New object("name";"Winch";"dateHired";!16-05-2018!;"age";36))
$c.push(New object("name";"Sterling";"dateHired";!10-5-1999!;"age";Null))
$c.push(New object("name";"Mark";"dateHired";!01-01-2002!))
```

Este ejemplo devuelve la personas cuyo nombre contiene "in":

```
$col:=$c.query("name = :1";"@in@")
 //$col=[{name:Winch...},{name:Sterling...}]
```

Este ejemplo devuelve las personas cuyo nombre no comienza con una cadena de una variable (ingresada por el usuario, por ejemplo):

```
$col:=$c.query("name # :1";$aString+"@")
 //if $astring="W"
 //$col=[{name:Smith...},{name:Sterling...},{name:Mark...}]
```

Este ejemplo devuelve las personas cuya edad no se conoce (propiedad definida como nula o indefinida):

```
$col:=$c.query("age=null") //placeholders not allowed with "null"
 //$col=[{name:Wesson...},{name:Sterling...},{name:Mark...}]
```

Este ejemplo devuelve las personas contratadas hace más de 90 días:

```
$col:=$c.query("dateHired < :1";(Current date-90))
 //$col=[{name:Smith...},{name:Sterling...},{name:Mark...}] if today is 01/10/2018
```

Nota: este último ejemplo requiere que esté marcada la opción de compatibilidad "Utilizar el tipo de fecha en lugar del formato de fecha ISO en objetos" (ver **Página Compatibilidad**).

collection.reduce( )

collection.reduce (nomMet {; ValorInicial}{; param}{; param2 ; ... ; paramN}) -> Resultado

Parámetro	Tipo		Descripción
nomMet	Texto	➡	Nombre de la función a llamar para procesar los elementos de la colección
ValorInicial	Texto, Número, Objeto, Collection, Fecha, Booleano	➡	Valor a utilizar como primer argumento para la primera llamada de nomMet
param	Expresión	➡	Parámetro(s) a pasar a nomMet
Resultado	Booleano, Collection, Fecha, Número, Objeto, Texto	↩	Resultado del valor acumulador

Descripción

El método **collection.reduce( )** aplica el método de retrollamada *nomMet* contra un acumulador y cada elemento de la colección (de izquierda a derecha) para reducirlo a un único valor.

Nota: este método no modifica la colección original.

En *nomMet*, pase el nombre del método a utilizar para evaluar los elementos de la colección, junto con su(s) parámetro(s) en *param* (opcional). *nomMet* toma cada elemento de colección y realiza toda operación deseada para acumular el resultado en *$1.accumulator*, que se devuelve en *$1.value*.

Puede pasar el valor para inicializar el acumulador en *valorInicial*. Si se omite, *$1.accumulator* comienza con *Undefined*.

nomMet recibe los siguientes parámetros:

- en *$1.value*: valor del elemento a procesar
- en *$2*: *param*
- en *$N*...: param2...paramN

nomMet define los siguientes parámetros:

- *$1.accumulator*: valor a modificar por la función y que es inicializado por *valorInicial*.
- *$1.stop* (booleano, opcional): **true** para detener la retrollamada del método. El valor devuelto es el último calculado.

Ejemplo 1

```
C_COLLECTION($c)
$c:=New collection(5;3;5;1;3;4;4;6;2;2)
$r:=$c.reduce("Multiply";1) //returns 86400
```

Con el siguiente método ***Multiply***:

```
If(Value type($1.value)=Is real)
   $1.accumulator:=$1.accumulator*$1.value
End if
```

Ejemplo 2

Este ejemplo permite reducir varios elementos de colección a uno solo:

```
C_COLLECTION($c;$r)
$c:=New collection
$c.push(New collection(0;1))
$c.push(New collection(2;3))
$c.push(New collection(4;5))
$c.push(New collection(6;7))
$r:=$c.reduce("Flatten") //$r=[0,1,2,3,4,5,6,7]
```

Con el siguiente método ***Flatten***:

```
If($1.accumulator=Null)
   $1.accumulator:=New collection
End if
$1.accumulator.combine($1.value)
```

collection.remove( )

collection.remove (posicIni {; cuantos}) -> Resultado

Parámetro	Tipo		Descripción
posicIni	Entero largo	➡	Elemento en el que comenzar la eliminación
cuantos	Entero largo	➡	Cantidad de elementos a eliminar, o 1 elemento si se omite
Resultado	Collection	⮌	Colección original sin elemento(s) eliminado(s)

Descripción

El método **collection.remove( )** elimina uno o más elementos de la posición *posicInicial* en la colección y devuelve la colección editada.

Nota: este método modifica la colección original.

En *posicInicial*, pase la posición donde desea que se elimine el elemento de la colección. **Advertencia**: tenga en cuenta que los elementos de la colección se numeran desde 0. Si *posicInicial* es mayor que la longitud de la colección, la posición inicial real se establecerá para la longitud de la colección.

- Si *posicInicial* < 0, se vuelve a calcular como *posicInicial :=posicInicial +longitud* (se considera como punto de partida del calculo de la posición).
- Si el valor calculado < 0, *posicInicial* toma el valor 0.
- Si el valor calculado > longitud de la colección, *posicInicial* toma como valor la longitud de la colección.

En *cuantos*, pase la cantidad de elementos a eliminar de *posicInicial*. Si *cuantos* no está especificado, entonces se elimina un elemento.

Si intenta eliminar un elemento de una colección vacía, el método no hace nada (no se genera ningún error).

Ejemplo

```
C_COLLECTION($col)
$col:=New collection("a";"b";"c";"d";"e";"f";"g";"h")
$col.remove(3) // $col=["a","b","c","e","f","g","h"]
$col.remove(3;2) // $col=["a","b","c","g","h"]
$col.remove(-8;1) // $col=["b","c","g","h"]
$col.remove(-3;1) // $col=["b","g","h"]
```

collection.resize( )

collection.resize (tam {; valorDefecto}) -> Resultado

Parámetro	Tipo		Descripción
tam	Entero largo	➡	Nuevo tamaño de la colección
valorDefecto	Número, Texto, Objeto, Collection, Fecha, Booleano	➡	Valor por defecto para llenar nuevos elementos
Resultado	Collection	↩	Colección original redimensionada

Descripción

El método **collection.resize( )** define la longitud de la colección para el nuevo *tam* especificado y devuelve la colección redimensionada.

Nota: este método modifica la colección original.

- Si *tam* < longitud de la colección, los elementos excedentes se eliminan de la colección.
- Si *tam* > longitud de la colección, la longitud de la colección se aumenta a *tam*.
  Por defecto, los nuevos elementos se llenan con valores **null**. Puede especificar el valor para llenar en los elementos añadidos utilizando el parámetro *valorDefecto*.

Ejemplo

```
C_COLLECTION($c)
$c:=New collection
$c.resize(10) // $c=[null,null,null,null,null,null,null,null,null,null]

$c:=New collection
$c.resize(10;0) // $c=[0,0,0,0,0,0,0,0,0,0]

$c:=New collection(1;2;3;4;5)
$c.resize(10;New object("name";"X")) //$c=[1,2,3,4,5,{name:X},{name:X},{name:X},{name:X},{name:X}]

$c:=New collection(1;2;3;4;5)
$c.resize(2) //$c=[1,2]
```

collection.reverse( )

collection.reverse () -> Resultado

Parámetro	Tipo		Descripción
Resultado	Collection		Copia invertida de la colección

Descripción

El método **collection.reverse( )** devuelve una copia completa de la colección con todos sus elementos en orden inverso. Si la colección original es una colección compartida, la colección devuelta también es una colección compartida.

Nota: este método no modifica la colección original.

Ejemplo

```
C_COLLECTION($c)
$c:=New collection(1;3;5;2;4;6)
$c2:=$c.reverse() //$c2=[6,4,2,5,3,1]
```

collection.shift( )

collection.shift () -> Resultado

Parámetro	Tipo		Descripción
Resultado	Collection, Fecha, Número, Objeto, Texto	➲	Primer elemento de la colección

Descripción

El método **collection.shift( )** elimina el primer elemento de la colección y lo devuelve como el resultado de la función.
Nota: este método modifica la colección original.
Si la colección está vacía, este método no hace nada.

Ejemplo

```
C_COLLECTION($c)
$c:=New collection(1;2;4;5;6;7;8)
$val:=$c.shift()
 // $val=1
 // $c=[2,4,5,6,7,8]
```

collection.slice( )

collection.slice (comenzarDesde {; fin}) -> Resultado

Parámetro	Tipo		Descripción
comenzarDesde	Entero largo	➡	Índice de inicio (incluido)
fin	Entero largo	➡	Índice de fin (no incluido)
Resultado	Collection	⮌	Nueva colección que contiene elementos cortados (copia superficial)

Descripción

El método **collection.slice( )** devuelve una parte de una colección en una nueva colección, seleccionada desde el índice *comenzarDesde* hasta el índice *fin* (*fin* no incluido). Este método devuelve una copia superficial de la colección. Si la colección original es una colección compartida, la colección devuelta también es una colección compartida.

Nota: este método no modifica la colección original.

La colección devuelta contiene el elemento especificado por *comenzarDesde* y todos los elementos posteriores hasta, pero sin incluir, el elemento especificado por *fin*.
Si solo se especifica el parámetro *comenzarDesde*, la colección devuelta contiene todos los elementos desde *comenzarDesde* hasta el último elemento de la colección original..

- Si *comenzarDesde* < 0, se vuelve a calcular como *comenzarDesde :=comenzarDesde +length* (se considera como punto de inicio del calculo de la posición).
- Si el valor calculado < 0, *comenzarDesde* toma el valor 0.
- Si *fin* < 0, se recalcula como *fin*:=*fin*+*length*.
- Si *fin* < *comenzarDesde* (valores pasados o calculados), el método no hace nada.

Ejemplo

```
C_COLLECTION($c;$nc)
$c:=New collection(1;2;3;4;5)
$nc:=$c.slice(0;3) //$nc=[1,2,3]
$nc:=$c.slice(3) //$nc=[4,5]
$nc:=$c.slice(1;-1) //$nc=[2,3,4]
$nc:=$c.slice(-3;-2) //$nc=[3]
```

collection.some( )

collection.some ({posicionInicial ;} nomMet {; param {; param2 ; ... ; paramN}}) -> Resultado

Parámetro	Tipo		Descripción
posicionInicial	Entero largo	➡	Elemento a partir del cual comenzar la evaluación
nomMet	Texto	➡	Nombre del método a llamar para la evaluación
param	Expresión	➡	Parámetro(s) a pasar a nomMet
Resultado	Booleano	⮌	True si la evaluación de al menos un elemento devuelve true

Descripción

El método **collection.some( )** devuelve **true** si al menos un elemento de la colección ha sido evaluado como true para la prueba implementada en el método *nomMet*.

De forma predeterminada, **collection.some( )** prueba toda la colección. Opcionalmente, puede pasar el índice del elemento desde el que comienza la prueba en *posicionInicial*.

- Si *posicionInicial*>= la longitud de la colección, **False** se devuelve, lo que significa que la colección no se prueba.
- Si *posicionInicial*<0, el fin de la colección se considera como punto de inicio del cálculo de la posición.
- Si *posicionInicial*=0, se busca en toda la colección (por defecto).

En *nonMet*, pase el nombre del método a usar para evaluar los elementos de la colección, junto con su(s) parámetro(s) en *param* (opcional). *nonMet* puede realizar cualquier prueba, con o sin los parámetros. Este método recibe un parámetro *Object* en $1 y debe definir *$1.result* como **true** para cada elemento que cumple la prueba.

nonMet recibe los siguientes parámetros:

- en *$1.value*: valor del elemento a evaluar
- en *$2*: *param*
- en *$N*...: param2...paramN

nonMet establece los siguientes parámetros:

- *$1.result* (booleano): **true** si la evaluación del valor del elemento es exitosa, de lo contrario, **false**.
- *$1.stop* (booleano, opcional): **true** para detener la retrollamada del método. El valor devuelto es el último calculado.

En todo caso, en el punto donde el método **collection.some( )** encuentra el primer elemento de colección que devuelve **true** en *$1.result*, deja de llamar a *nonMet* y devuelve **true**.

Ejemplo

```
C_COLLECTION($c)
C_BOOLEAN($b)
$c:=New collection
$c.push(-5;-3;-1;-4;-6;-2)
$b:=$c.some("NumberGreaterThan0")  // devuelve false
$c.push(1)
$b:=$c.some("NumberGreaterThan0")  // devuelve true

$c:=New collection
$c.push(1;-5;-3;-1;-4;-6;-2)
$b:=$c.some("NumberGreaterThan0")  //$b=true
$b:=$c.some(1;"NumberGreaterThan0")  //$b=false
```

Con el siguiente método ***NumberGreaterThan0***:

```
$1.result:=$1.value>0
```

collection.sort( )

collection.sort ({nomMet {; extraParam}{; extraParam2 ; ... ; extraParamN}}) -> Resultado

Parámetro	Tipo		Descripción
nomMet	Texto	➡	Nombre del método utilizado para especificar el orden de clasificación
extraParam	Expresión	➡	Parámetro(s) para el método
Resultado	Collection	⮌	Copia ordenada de la colección (copia superficial)

Descripción

El método **collection.sort( )** ordena los elementos de la colección y devuelve una nueva colección ordenad. Este método devuelve una *copia superficial*, lo que significa que los objetos o colecciones en ambas colecciones comparten la misma referencia.

Nota: este método modifica la colección original.

Si se llama a **collection.sort( )** sin parámetros, solo se clasifican los valores escalares (número, texto, fecha, booleanos). Los elementos se ordenan por defecto en orden ascendente, de acuerdo con su tipo.

Si desea ordenar los elementos de la colección en otro orden u ordenar cualquier tipo de elemento, debe ofrecer en *nomMet* un método de comparación que compare dos valores y devuelva **true** en *$1.result* si el primer valor es menor que el segundo valor. Puede ofrecer parámetros adicionales a *nomMet* si es necesario.

- *nomMet* recibirá los siguientes parámetros:
  - $1 (objeto), donde:
    - *$1.value* (todo tipo): valor del primer elemento a ser comparado
    - *$1.value2* (todo tipo): valor del segundo elemento a ser comparado
  - $2...$N (todo tipo): parámetros adicionales
- *nomMet* define el siguiente parámetro:
  - *$1.result* (booleano): **true** si *$1.value* < *$1.value2*, de lo contrario false

Si la colección contiene elementos de diferentes tipos, primero se agrupan por tipo y se ordenan después. Los tipos se devuelven en el siguiente orden:

1. null
2. booleanos
3. cadenas
4. números
5. objetos
6. colecciones
7. fechas

Ejemplo 1

```
C_COLLECTION($col)
$col:=New collection("Tom";5;"Mary";3;"Henry";1;"Jane";4;"Artie";6;"Chip";2)
$col2:=$col.sort() // $col2=["Artie","Chip","Henry","Jane","Mary","Tom",1,2,3,4,5,6]
```

Ejemplo 2

```
C_COLLECTION($col)
$col:=New collection(10;20)
$col2:=$col.push(5;3;1;4;6;2).sort() //$col2=[1,2,3,4,5,6,10,20]
```

Ejemplo 3

```
C_COLLECTION($col)
$col:=New collection(33;4;66;1111;222)
$col2:=$col.sort() //orden numérico: [4,33,66,222,1111]
$col3:=$col.sort("numberOrder") //orden alfabético: [1111,222,33,4,66]
```

```
//Método proyecto numberOrder
C_OBJECT($1)
$1.result:=String($1.value)<String($1.value2)
```

collection.sum( )

collection.sum ({rutaProp}) -> Resultado

Parámetro	Tipo		Descripción
rutaProp	Texto	➡	Ruta de la propiedad del objeto que se utilizará para el cálculo
Resultado	Real	⮌	Suma de valores de la colección

Descripción

El método **collection.sum( )** devuelve la suma de todos los valores numéricos de la colección.
Solo se tienen en cuenta elementos numéricos para el cálculo (se ignoran otros tipos de elementos).
Si la colección contiene objetos, pase el parámetro *rutaProp* para indicar la propiedad del objeto a tener en cuenta.
collection.sum( ) devuelve 0 si:

- la colección está vacía,
- la colección no contiene elementos numéricos,
- *rutaProp* no se encuentra en la colección.

Ejemplo 1

```
C_COLLECTION($col)
$col:=New collection(10;20;"Monday";True;2)
$vsum:=$col.sum() //32
```

Ejemplo 2

```
C_COLLECTION($col)
$col:=New collection
$col.push(New object("name";"Smith";"salary";10000))
$col.push(New object("name";"Wesson";"salary";50000))
$col.push(New object("name";"Gross";"salary";10500,5))
$vSum:=$col.sum("salary") //$vSum=70500,5
```

collection.unshift( )

collection.unshift (valor {; valor2 ; ... ; valorN}) -> Resultado

Parámetro	Tipo		Descripción
valor	Expresión	➡	Valor(es) a insertar al comienzo de la colección
Resultado	Collection	⮌	Colección que contiene el(los) elemento(s) añadido(s)

Descripción

La función **collection.unshift( )** inserta el (los) valor(es) dado(s) al comienzo de la colección y devuelve la colección modificada.

Nota: este comando modifica la colección original.

Si se pasan varios valores, se insertan todos a la vez, lo que significa que aparecen en la colección resultante en el mismo orden que en la lista de argumentos.

Ejemplo

```
C_COLLECTION($c)
$c:=New collection(1;2)
$c.unshift(4) // $c=[4,1,2]
$c.unshift(5) //$c=[5,4,1,2]
$c.unshift(6;7) // $c=[6,7,5,4,1,2]
```

New collection

New collection {(valor {; valor2 ; ... ; valorN})} -> Resultado

Parámetro	Tipo		Descripción
valor	Número, Texto, Fecha, Objeto, Collection, Puntero	➡	Valor de la colección
Resultado	Collection	⮌	Nueva colección

Descripción

El comando **New collection** crea una nueva colección vacía o prellena y devuelve su referencia.

Si no pasa ningún parámetro, **New collection** crea una colección vacía y devuelve su referencia.

Debe asignar la referencia devuelta a una variable 4D declarada con **C_COLLECTION**.
Nota: tenga en cuenta que **C_COLLECTION** declara una variable de tipo *Colección* pero no crea ninguna colección.

Opcionalmente, puede prellenar la nueva colección pasando uno o varios *valores* como parámetro(s).

De lo contrario, puede agregar o modificar elementos posteriormente a través de la asignación de la notación de objetos. Por ejemplo:

```
myCol[10]:="My new element"
```

Si el nuevo índice de elementos está más allá del último elemento existente de la colección, la colección se redimensiona automáticamente y todos los elementos intermediarios nuevos obtienen el valor **null**.

Nota: para más información sobre la notación de objetos, consulte la sección **Uso de la notación objeto**.

Puede pasar todo número de valores de los tipo soportados (número, texto, fecha, puntero, objeto, colección...). A diferencia de los arrays, las colecciones pueden mezclar datos de diferentes tipos.

Debe prestar atención a los siguientes problemas de conversión:

- Si pasa un puntero, se mantiene como es; Se evalúa mediante el comando **JSON Stringify**
- Las fechas se almacenan en el formato de fecha "aaaa-mm-dd" o de cadena al formtao "AAAA-MM-DDTHH:mm:ss.SSSZ" en función del parámetro actual relativo al almacenamiento de fechas en objetos (ver **Página Compatibilidad**). Al convertir fechas 4D en texto, antes de almacenarlas en la colección, de forma predeterminada, el programa toma en cuenta la zona horaria local. Puede modificar este comportamiento utilizando el selector Dates inside objects del comando **SET DATABASE PARAMETER**.
- Si pasa un tiempo, se almacena como un número de milisegundos (Real).

Ejemplo 1

Usted desea crear una nueva colección vacía y asignarla a una variable colección 4D:

```
C_COLLECTION($myCol)
$myCol:=New collection
 //$myCol=[]
```

Ejemplo 2

Usted desea crear una colección precargada:

```
C_COLLECTION($filledColl)
$filledColl:=New collection(33;"mike";"november";->myPtr;Current date)
 //$filledColl=[33,"mike","november","->myPtr","2017-03-28T22:00:00.000Z"]
```

Ejemplo 3

Usted crea una nueva colección y luego agrega un nuevo elemento:

```
C_COLLECTION($coll)
$coll:=New collection("a";"b";"c")
 //$coll=["a","b","c"]
$coll[9]:="z"  //add a 10th element with value "z"
$vcolSize:=$coll.length  //10
 //$coll=["a","b","c",null,null,null,null,null,null,"z"]
```

Nota: este ejemplo requiere que la notación de objeto esté activada en la base de datos (ver el párrafo **Notación objeto**).

New shared collection

New shared collection {(valor {; valor2 ; ... ; valorN})} -> Resultado

Parámetro	Tipo		Descripción
valor		➡	Valor(es) de la colección compartida
Resultado	Collection	➲	Nueva colección compartida

Descripción

El comando **New shared collection** crea una nueva colección compartida vacía o prellenada y devuelve su referencia. Agregar un elemento a esta colección debe estar rodeado por la estructura **Use...End use**, de lo contrario, se devuelve un error. Sin embargo, es posible leer un elemento fuera de una estructura **Use...End use**.

Nota: para más información sobre *colecciones compartidas*, consulte la página **Objetos y colecciones compartidos**.

Si no pasa ningún parámetro, **New shared collection** crea una colección compartida vacía y devuelve su referencia.

Debe asignar la referencia devuelta a una variable 4D declarada con el comando **C_COLLECTION**.

Nota: tenga en cuenta que **C_COLLECTION** declara una variable de tipo *Collection*, pero no crea una colección.

Opcionalmente, puede llenar previamente la nueva colección compartida pasando uno o varios *valores* como parámetro(s). De lo contrario, puede agregar o modificar elementos posteriormente a través de la asignación de notación de objeto (ver ejemplo).

Si el índice del nuevo elemento está más allá del último elemento existente de la colección compartida, la colección se redimensionará automáticamente y todos los nuevos elementos intermediarios obtendrán el valor **null**.

- Puede número (real, entero largo...). Los valores numéricos siempre se almacenan como reales.
- texto
- booleano
- fecha
- hora (almacenado como número de milisegundos - real)
- nulo
- objeto compartido(*)
- colección compartida(*)

Nota: a diferencia de las colecciones estándar (no compartidas), las colecciones compartidas no admiten imágenes, punteros y objetos o colecciones que no se compartan.

(*)Cuando un objeto compartido o colección se agrega a una colección compartida, comparten el mismo identificador de *bloqueo*. Para más información sobre este punto, consulte la sección **Identificador de bloqueo**.

Ejemplo

```
$mySharedCol:=New shared collection("alpha";"omega")
Use($mySharedCol)
    $mySharedCol[1]:="beta"
End use
```

Compilador

- Comandos del Compilador
- Utilización de directivas de compilación
- Guía de declaración
- Detalles de sintaxis
- Consejos de optimización
- Mensajes de error
- C_BLOB
- C_BOOLEAN
- C_COLLECTION
- C_DATE
- C_LONGINT
- C_OBJECT
- C_PICTURE
- C_POINTER
- C_REAL
- C_TEXT
- C_TIME
- IDLE
- *_o_C_GRAPH*
- *_o_C_INTEGER*
- *_o_C_STRING*

Comandos del Compilador

El compilador integrado de 4D traduce sus aplicaciones de bases de datos en instrucciones de nivel de ensamblador. Las ventajas del compilador son:

- **Velocidad:** su base de datos puede se ejecuta de 3 a 1 000 veces más rápido.
- **Verificación del código:** su aplicación de base de datos se revisa para tener consistencia en el código. Se detectan los conflictos de lógica y de sintaxis.
- **Protección:** una vez compilada su base de datos, puede borrar el código interpretado. Entonces, la base compilada tiene las mismas funcionalidades que la base original, excepto que la estructura y los métodos no pueden ser visualizados o modificados, deliberadamente o inadvertidamente.
- **Aplicaciones independientes "doble-clic":** las bases compiladas también pueden transformarse en aplicaciones independientes (archivos .EXE) con su propio icono.
- **Ejecución en modo apropiativo**: sólo el código compilado puedes ser ejecutado en un proceso apropiativo.

Para una descripción del funcionamiento del compilador, consulte el Manual de Diseño.

Los comandos de este tema están relacionados con el uso del compilador. Ellos le permiten normalizar los tipos de datos a través de su base de datos. El comando **IDLE** se utiliza específicamente en las bases compiladas.

C_BLOB	**C_REAL**	**C_TEXT**
C_BOOLEAN	**C_LONGINT**	**C_DATE**
C_POINTER	**C_PICTURE**	**C_TIME**
C_OBJECT	**IDLE**	

Nota de compatibilidad: los comandos obsoletos **_o_C_GRAPH**, **_o_C_INTEGER** y **_o_C_STRING** ya no deben utilizarse.

Estos comandos, excepto **IDLE**, declaran a las variables y les asignan un tipo de datos específico. La declaración de variables resuelve ambigüedades relacionadas con el tipo de dato de las variables. Cuando una variable no es declarada con uno de estos comandos, el compilador intenta determinar el tipo de dato de la variable. Es difícil para el compilador determinar los tipos de datos de una variable utilizada en un formulario. Por lo tanto, es particularmente importante utilizar estos comandos para declarar las variables utilizadas en formularios.

Nota: para ahorrar tiempo, puede utilizar la opción de generación y actualización de métodos de digitación (llamada "Métodos compilador") que se encuentra en la ventana del compilador. Esta opción crea automáticamente métodos de declaración que evalúan y asignan un tipo para todas las variables utilizadas en la base.

Los **Arrays** son variables que deben respetar las mismas reglas que las variables estándar con respecto a la compilación. Los comandos de declaración de los arrays se agrupan en el tema "**Arrays**".

Reglas generales de escritura de código a compilar

- No debe dar el mismo nombre a más de un método o variable. No puede tener un método con el mismo nombre de una variable.
- No se permite la indirección de variables como se utilizó en 4D versión 1. No puede utilizar indirección alfa, con el símbolo sección (§), para referenciar variables indirectamente. No puede utilizar indirección numérica, con corchetes ({...}). Los corchetes sólo pueden ser utilizados cuando se accede a elementos específicos de un Array que ha sido declarado. Sin embargo, puede utilizar indirección de parámetros.
- No puede cambiar el tipo de dato de una variable o un Array.
- No puede convertir un Array de una dimensión en un Array bidimensional y viceversa.
- No puede modificar la longitud de una variable cadena ni de los elementos en arrays alfanuméricos.
- Aunque el compilador deduce el tipo de variable por usted, cuando los tipos de datos sean ambiguos, como en un formulario, debe especificar el tipo de dato utilizando las directivas de compilación.
- Otra razón para declarar explícitamente los tipos de variables es la optimización de su código. Esta regla aplica especialmente a cualquier variable utilizada como contador. Utilice variables de tipo Entero largo para asegurar un máximo desempeño.
- Para borrar una variable (inicializarla en un valor nulo), utilice el comando **CLEAR VARIABLE** con el nombre de la variable. No utilice una cadena para representar el nombre de la variable con el comando **CLEAR VARIABLE**.
- La función **Undefined** siempre devolverá **False**. Las variables siempre estarán definidas.
- Las operaciones numéricas efectuadas en variables de tipo Entero largo y Entero son generalmente mucho más rápidas que las operaciones efectuadas en variables de tipo numérico (real).
- Si seleccionó la propiedad "Puede ejecutarse en un proceso apropiativo" para el método, el código no debe llamar a los comandos hilo no seguros o a otros métodos hilo no seguro.

Estos principios se detallan en las siguientes secciones:

- **Utilización de directivas de compilación**, explica cuándo y dónde escribir directivas de compilación,
- **Guía de declaración**, describe los diferentes tipos de conflictos que pueden ocurrir durante la compilación de bases de datos 4D,
- **Detalles de sintaxis**, ofrece información adicional relacionada con varios comandos 4D,
- **Consejos de optimización** , ofrece consejos para acelerar la ejecución de aplicaciones en modo compilado.

- **Escribir un método hilo seguro**.

Ejemplo 1

Las siguientes son algunas declaraciones de variables estándar para el compilador:

```
C_BLOB(vxMyBlob) // La variable proceso vxMBlob está declarada como una variable de tipo BLOB
C_BOOLEAN(<>OnWindows) // La variable interproceso <>OnWindows está declarada como una variable de tipo Booleano
C_DATE($vdCurDate) // La variable local $vdCurDate está declarada como una variable de tipo Fecha
C_LONGINT(vg1;vg2;vg3) // Las 3 variables de proceso vg1, vg2 y vg3 están declaradas como variables de tipo entero largo
```

Ejemplo 2

En el siguiente ejemplo, el método proyecto **UnMetodoEntreOtros** declara 3 parámetros:

```
// Método proyecto UnMetodoEntreOtros
// UnMetodoEntreOtros ( Real ; Date { ; Long } )
// UnMetodoEntreOtros ( Amount ; Date{ ; Ratio } )

C_REAL($1) // 1er parámetro es de tipo Real
C_DATE($2) // 2do parámetro es de tipo Entero
C_LONGINT($3) // 3er parámetro es de tipo Entero largo

// ...
```

Ejemplo 3

En el siguiente ejemplo, el método proyecto **Mayusculas** acepta un parámetro de tipo texto y devuelve un texto:

```
// Método proyecto Mayusculas
// Mayusculas ( Texto) -> Texto
// Mayusculas ( Cadena fuente ) -> Cadena en mayúsculas

C_TEXT($0;$1)
$0:=Uppercase(Substring($1;1;1))+Lowercase(Substring($1;2))
```

Ejemplo 4

En el siguiente ejemplo, el método de proyecto **ENVIAR PAQUETES** acepta un parámetro tipo Hora seguido por una variable de parámetros de tipo Texto:

```
` Método de proyecto ENVIAR PAQUETES
` ENVIAR PAQUETES ( Hora; Texto { ; Texot2... ; TextoN } )
` ENVIAR PAQUETES ( docRef ; Data { ; Data2... ; DataN } )

C_TIME($1)
C_TEXT($${2})
C_LONGINT($vlPacket)

For($vlPacket;2;Count parameters)
   SEND PACKET($1;$${$vlPacket})
End for
```

Ejemplo 5

En el siguiente ejemplo, el método de proyecto ***COMPILER_Param_Predeclare28*** predeclara la sintaxis de otros métodos proyecto para el compilador:

```
// Método proyecto COMPILER_Param_Predeclare28

C_REAL(UnMetodoEntreOtros;$1) // UnMetodoEntreOtros( Real ; Integer { ; Long } )
C_DATE(UnMetodoEntreOtros;$2) // ...
C_LONGINT(UnMetodoEntreOtros;$3) // ...
C_TEXT(Capitalize;$0;$1) // Capitalize ( Text) -> Text
C_TIME(SEND PACKETS;$1) // SEND PACKETS ( Time ; Text { ; Text2... ; TextN } )
C_TEXT(SEND PACKETS;$${2}) // ...
```

Utilización de directivas de compilación

Tipos de variables

4D tiene tres categorías de variables:

- Variables locales,
- Variables proceso,
- Variables interproceso.

Para mayor información sobre este punto, consulte la sección **Variables**. Las variables proceso e interproceso son estructuralmente de la misma naturaleza del compilador.

Como el compilador no puede determinar el proceso en el cual la variable será utilizada, las variables proceso deben utilizarse con más cuidado que las variables interproceso. Todas las variables proceso se duplican sistemáticamente cuando comienza un nuevo proceso. Una variable proceso puede tener un valor diferente en cada proceso, pero tiene el mismo tipo en toda la base.

Tipos de variables

Todas las variables tienen un tipo. Como se describe en la sección **Tipos de datos**, hay varios tipos de variables:

Booleano
Alfanumérico (o cadena fija)
Fecha
Entero
Entero largo
Hora
Imagen
Número (o Real)
Puntero
Texto
BLOB
Objeto
Colección

Para las variables de tipo Array, dispone de los siguientes tipos:

Array Booleano
Array Alfa
Array Fecha
Array Entero
Array Entero largo
Array Imagen
Array Real
Array Hora
Array Objeto
Array Puntero
Array BLOB
Array Texto

Notas de compatibilidad:
- El tipo "objeto" está disponible a partir de 4D v14.
- El tipo Colección está disponible desde 4D v16 R4
- El tipo anterior Alfa (cadena de longitud fija) ya no se utiliza más para las variables. En el código existente, se redirigen automáticamente al tipo Texto. Los tipos anteriores Entero y Gráfico ya no se utilizan para variables. En el código existente, se redirigen automáticamente a los tipos Entero largo.

Creación de la tabla de símbolos

En modo interpretado, una variable puede tener más de un tipo de datos. Esto es posible porque el código es interpretado en lugar de compilado. 4D interpreta cada instrucción por separado y comprende su contexto. Cuando trabaja en modo compilado, la situación es diferente. Mientras la interpretación se realiza línea por línea, el proceso de compilación mira a una base en su globalidad.

La manera de operar el compilador es la siguiente:

- El compilador analiza sistemáticamente los objetos en la base. Los objetos son los métodos base, proyecto, formulario, trigger y objeto.
- El compilador escanea los objetos para determinar el tipo de dato de cada variable utilizada en la base y genera la tabla de variables y métodos (la tabla símbolo).
- Una vez establecidos los tipos de datos de todas las variables, el compilador traduce (compila) la base. Sin embargo, no puede compilar la base a menos que pueda determinar el tipo de dato de cada variable.

Si el compilador encuentra el mismo nombre de variable en dos tipos diferentes de datos, no tiene ninguna razón para favorecer a uno de ellos. En otras palabras, para clasificar un objeto y darle una dirección de memoria, el compilador debe saber la identidad precisa del objeto (es decir, su nombre y su tipo). El tipo le permite al compilador determinar el tamaño. Para cada base compilada, el compilador crea un mapa que lista, para cada variable, su nombre (o identificador), su ubicación (o dirección de memoria), y el espacio que ocupa (indicado por su tipo). Este mapa se llama la tabla de símbolos. Una opción en las preferencias le permite generar o no esta tabla en forma de archivo durante la compilación.

Este mapa también se utiliza para la generación automática de métodos compilador.

Variables declaradas por el compilador

Si quiere que el compilador verifique los tipos de sus variables o que declare las variables él mismo, es fácil colocar una directiva de compilación para este propósito. Puede elegir entre dos posibilidades diferentes, dependiendo de sus métodos de trabajo:

- Utilice la directiva en el método en el cual aparece la variable por primera vez, dependiendo si es una variable local, proceso o interproceso. Asegúrese de utilizar la directiva la primera vez que utilice la variable, en el primer método a ejecutar. Recuerde que durante la compilación, el compilador toma los métodos en el orden de su creación en 4D, y no en el orden en el cual aparecen en el Explorador.
- O si usted es sistemático, agrupe todos las variables proceso e interproceso con las diferentes directivas de compilación en el **Método base On Startup** o en un método llamado por el **Método de base de datpos On Startup**.
  Para las variables locales, agrupe las directivas al comienzo del método en el cual aparecen.

Valores por defecto

Cuando las variables se digitan por medio de una directiva de compilación, reciben un valor por defecto, que se conserva durante la sesión, siempre y cuando no se hayan asignado.

El valor por defecto depende del tipo y la categoría variable, del contexto de ejecución (interpretado o compilado), así como, para el modo compilado, la6s opciones de compilación definidas en la **Página Compilador** de las Propiedades de la base:

- Las variables de proceso e interprocesos se definen siempre "en cero" (lo que significa, según el caso, "0", una cadena vacía, un Blob vacío, un puntero Nil, una fecha vacía (00-00-00), etc.)
- Las variables locales se definen:
  - en modo interpretado: en cero
  - en modo compilado, dependiendo de la opción **Inicializar las variables locales** de las Propiedades de la base:
    - en cero cuando "en cero" esté seleccionado,
    - en un valor aleatorio definido cuando "en un valor aleatorio" se elige (0x72677267 para los números y horas, siempre es True para booleanos, lo mismo que "en cero" para los demás),
    - en un valor aleatorio (para los números) cuando "no" esté seleccionado.

La siguiente tabla ilustra estos valores por defecto:

Tipo	Interproceso	Proceso	Local interpretado	Local compilado "en cero"	Local compilado "aleatorio"	Local compilado "no"
Booleeno	False	False	False	False	True	True (varía)
Fecha	00-00-00	00-00-00	00-00-00	00-00-00	00-00-00	00-00-00
Entero largo	0	0	0	0	1919382119	909540880 (varía)
Gráfico	0	0	0	0	1919382119	775946656 (varía)
Hora	00:00:00	00:00:00	00:00:00	00:00:00	533161:41:59	249345:34:24 (varía)
Imagen	picture size=0	picture size=0	picture size=0	picture size=0	picture size=0	picture size=0
Real	0	0	0	0	1.250753659382e+243	1.972748538022e-217 (varía)
Puntero	Nil=true	Nil=true	Nil=true	Nil=true	Nil=true	Nil=true
Text	""	""	""	""	""	""
Blob	Blob size=0	Blob size=0	Blob size=0	Blob size=0	Blob size=0	Blob size=0
Objeto	nulo	nulo	nulo	nulo	nulo	nulo
Colección	nulo	nulo	nulo	nulo	nulo	nulo

¿Cuándo utilizar directivas de compilación?

Las directivas de compilación son útiles en dos casos:
• cuando el compilador no puede determinar el tipo de dato de una variable por su contexto,
• cuando usted no quiere que el compilador determine el tipo de la variable por su uso.
Adicionalmente, utilizar directivas de compilación le permite reducir el tiempo de compilación.

Casos de ambigüedad

Algunas veces el compilador no puede determinar el tipo de dato de una variable y genera un mensaje de error.
Hay tres causas principales que pueden evitar que el compilador determine el tipo de dato: tipos de datos múltiples, ambigüedad sobre una deducción forzada y la imposibilidad de determinar un tipo.

- Tipos de datos múltiples
  Si una variable tiene diferentes tipos en diferentes instrucciones en la base, el compilador genera un error que es fácil de solucionar.
  El compilador selecciona la primera variable que encuentra y le asigna su tipo de dato a la siguiente ocurrencia de la variable con el mismo nombre pero un tipo diferente.

  Este es un ejemplo sencillo:

en un método A,

```
Variable:=True
```

en un método B,

```
Variable:="La luna es verde"
```

Si el método A se compila antes que el método B, el compilador considera que **Variable:="La luna es verde"** es un cambio de tipo de una variable encontrada anteriormente. El compilador le notifica que ha ocurrido un cambio de tipo. Esto genera un error que debe corregir. En la mayoría de los casos, el problema puede solucionarse renombrando la segunda ocurrencia de la variable.

- Ambigüedad sobre una deducción forzada
  Algunas veces, por una secuencia, el compilador puede deducir que un tipo de objeto no es el apropiado. En este caso, debe explícitamente darle tipo a la variable con una directiva de compilación.

  Este es un ejemplo utilizando los valores por defecto para un objeto activo:
  En un formulario, puede asignar valores por defecto para los siguientes objetos: combo boxes, pestañas, listas y menús desplegables y áreas de desplazamiento utilizando el botón de edición de **Valores por defecto** (en el tema Fuente de datos de la Lista de propiedades) (para mayor información, consulte el Manual de Diseño). Los valores por defecto se cargan automáticamente en un Array cuyo nombre es el mismo del objeto.
  Si el objeto no se utiliza en un método, el compilador puede deducir el tipo, sin ambigüedad, como un array de texto.
  Sin embargo, si se debe efectuar una inicialización, la secuencia podría ser:

```
Case of
   :(Form event=On Load)
      MiPopUp:=2
      ...
End case
```

En este caso, aparece la ambigüedad, durante el análisis de los métodos, el compilador deducirá un tipo de dato Real para el objeto MiPopUp. En este caso, es necesario declarar explícitamente el Array en el método de formulario o en un método compilador:

```
Case of
   :(Form event=On Load)
      ARRAY TEXT(MiPopUp;2)
      MiPopUp:=2
      ...
End case
```

• Imposibilidad para determinar un tipo
Este caso puede surgir cuando una variable se utiliza sin haber sido declarada, en un contexto que no ofrece información sobre su tipo. En este caso, sólo una directiva de compilación puede guiar el compilador.
Este fenómeno ocurre principalmente en cuatro contextos:
- cuando utiliza punteros,
- cuando utiliza un comando con más de una sintaxis,
- cuando utiliza un comando con parámetros opcionales de diferentes tipos de datos,
- cuando utiliza un método 4D llamado vía un URL.

- Punteros
No se puede esperar que un puntero devuelva un tipo diferente al propio.
Considere la siguiente secuencia:

```
Var1:=5.2(1)
Puntero:=->Var1(2)
Var2:=Puntero->(3)
```

Aunque la línea (2) defina el tipo de variable apuntada por el puntero, el tipo de Var2 no se determina. En el momento de la compilación, el compilador puede reconocer un puntero, pero no tiene ningún medio de saber a que tipo de variable apunta. Por lo tanto no puede deducir el tipo de Var2. Se necesita una directiva de compilación, por ejemplo **C_REAL(Var2)**.

- Comandos de sintaxis múltiple
Cuando utiliza una variable asociada con la función **Year of**, la variable sólo puede ser de tipo Fecha, considerando la naturaleza de esta función. Sin embargo, las cosas no siempre son simples. Este es un ejemplo:
El comando **GET FIELD PROPERTIES** acepta dos sintaxis:
GET FIELD PROPERTIES(tablaNo;campoNo;Tipo;longitud;indice)
GET FIELD PROPERTIES(campoPuntero;Tipo;longitud;indice)

Cuando utiliza un comando de sintaxis múltiple, el compilador no puede adivinar la sintaxis y los comandos que usted escogío. Debe utilizar directivas de compilación para darle un tipo a las variables que se pasan al comando, si no tienen un tipo de acuerdo a su uso en la base.

- Comandos con parámetros opcionales de diferentes tipos
Cuando utiliza un comando que contiene varios parámetros opcionales de diferentes tipos, el compilador no puede determinar que parámetros opcionales han sido utilizados. Este es un ejemplo:
El comando **GET LIST ITEM** acepta dos parámetros opcionales; el primero es de tipo Entero largo y el segundo de tipo Booleano.
El comando puede ser utilizado:
GET LIST ITEM(lista;posición;itemRef;texto;sublista;expandido)
o como:
GET LIST ITEM(lista;posición;itemRef;texto;expandido)

Debe utilizar directivas de compilación para dar tipo a los parámetros opcionales pasados al comando, si no tienen tipos de acuerdo a su uso en la base.

- Métodos llamados vía URLs

Si escribe métodos 4D que deben llamarse vía un URL y si no utiliza $1 en el método, debe declarar explícitamente la variable texto $1 con la siguiente secuencia:

C_TEXT($1)

De hecho, el compilador no puede determinar que el método 4D se llamará vía un URL.

Reducción de tiempos de compilación

Si todas las variables utilizadas en la base se declaran explícitamente, no es necesario que el compilador revise los tipos. En este caso, puede fijar las opciones de manera que el compilador sólo ejecute la fase de traducción del método. Esto ahorra por lo menos el 50% del tiempo de compilación.

Caso de optimización

Usted puede acelerar sus métodos utilizando directivas de compilación. Para mayor información al respecto, consulte la sección **Consejos de optimización** . Para dar un ejemplo simple, imagine que necesita incrementar un contador utilizando una variable local. Si no declara la variable, el compilador asume que es de tipo Real. Si la declara como de tipo Entero largo, la ejecución de la base compilada será más eficiente. En un PC, por ejemplo, un Real ocupa 8 bytes en memoria, mientras que el tipo Entero largo, sólo utiliza 4 bytes. Incrementar un contador de 8 bytes obviamente toma más tiempo que aumentar un contador de 4 bytes.

¿Dónde ubicar sus directivas de compilación?

Las directivas de compilación se pueden manejar de dos formas diferentes, dependiendo de si quiere que el compilador verifique o no los tipos de sus variables.

Tipo de las variables

El compilador debe respetar el criterio de identificación de las variables.
Hay dos posibilidades:

- Si la variable no tiene tipo, el compilador se ocupará de eso automáticamente. Siempre que sea posible, siempre y cuando no haya ambigüedad, el compilador determina un tipo de variable dependiendo de su uso. Por ejemplo, si escribe:

```
V1:=True
```

el compilador determina que la variable V1 es de tipo booleano.

De la misma forma, si escribe:

```
V2:="Esta es una frase de ejemplo"
```

el compilador determina que V2 es una variable de tipo texto.

El compilador también es capaz de establecer el tipo de dato de una variable en casos menos fáciles:

```
V3:=V1  `V3 es del mismo tipo que V1
V4:=2*V2  `V4 es del mismo tipo que V2
```

El compilador también determina el tipo de datos de sus variables de acuerdo a los llamados a los comandos 4D y a sus métodos. Por ejemplo si pasa un parámetro tipo booleano y un parámetro tipo fecha a un método, el compilador asigna el tipo booleano y el tipo fecha a las variables locales $1 y $2 del método llamado.

Cuando el compilador determina el tipo de dato por deducción, a menos de que se indique de otra forma en las Propiedades de la base, el compilador asigna el tipo de dato más amplio posible. Por ejemplo, si escribe:

```
Número:=4
```

el compilador asigna el tipo de dato Real, aunque 4 sea un entero. En otras palabras, el compilador no descarta la posibilidad de que, bajo otras circunstancias, el valor de la variable pueda ser 4.5.

Si es conveniente darle a una variable un tipo Entero, Entero largo, o Alfa, puede hacer utilizando una directiva de compilación. Es una ventaja hacerlo, porque estos tipos de datos ocupan menos memoría y es mucho más rápido efectuar operaciones en ellos.

Si ya ha asignado un tipo a sus variables y está seguro de que su tipo coherente y completo, puede pedirle explícitamente al compilador no hacer nuevamente este trabajo, utilizando las Preferencias de compilación. En caso de que los tipos que asignó no estén completos, al momento de la compilación, el compilador devolverá errores solicitándole hacer las modificaciones necesarias.

- Los comandos de las directivas de compilación le permiten declarar explícitamente las variables utilizadas en sus base.

Se utilizan de la manera siguiente:

```
C_BOOLEAN(Var)
```

A través de tales directivas, usted le dice al compilador que cree una variable Var que será de tipo Booleano.
Cuando una aplicación incluye directivas de compilación, el compilador las detecta y evita hacer conjeturas.

Una directiva de compilación tiene prioridad sobre una deducción hecha de una asignación o uso.

Las variables declaradas por la directiva de compilación **C_INTEGER** en realidad son iguales a las declaradas por la directiva **C_LONGINT**. De hecho son enteros largos entre –2147483648 y +2147483647.

Variables declaradas por el desarrollador

Si no quiere que el compilador revise los tipos, debe darle un código para identificar las directivas de compilación.
La convención es la siguiente:
Las directivas de compilación de las variables proceso e interproceso y los parámetros deben ubicarse en uno o más métodos, cuyos nombres comiencen con la palabra clave **Compiler**.
Por defecto, el compilador le permite generar automáticamente cinco tipos de métodos Compiler, los cuales agrupan la directivas para las variables, arrays y los parámetros de los métodos (para mayor información sobre este punto, consulte el Manual de Diseño).

Nota: la sintaxis para la declaración de estos parámetros es:
Directiva (nombreMétodo;Parámetro). Esta sintaxis no es ejecutable en modo interpretado.

Parámetros particulares

- **Los parámetros recibidos por los métodos base**

Si estos parámetros no han sido declarados explícitamente, son declarados por el compilador. Sin embargo, si los declara, la declaración se debe hacer dentro de los métodos de base de datos.
La declaración de estos parámetros no puede hacerse en un método compilador.
Ejemplo: **Método de base de datos On Web Connection** recibe seis parámetros, $1 a $6, de tipo texto. Al comienzo del método, debe escribir: **C_TEXT($1;$2;$3;$4;$5;$6)**

- **Triggers**

El parámetro $0 (Entero largo), es el resultado de un trigger, es declarado por el compilador el parámetro no ha sido declarado explícitamente. Sin embargo, si quiere declararlo, debe hacerlo en el trigger.
La declaración de este parámetro no puede hacerse en un método compilador.

- **Los objetos aceptan el evento de formulario "On Drag Over"**

El parámetro $0 (Entero largo), es el resultado de un evento de formulario "On Drag Over", es declarado por el compilador si el parámetro no ha sido declarado explícitamente. Sin embargo, si quiere declararlo, debe hacerlo en el método de objeto.
La declaración de este parámetro no puede escribirse en un método compilador.

Nota: el compilador no inicializa el parámetro $0. De manera que tan pronto utilice el evento de formulario On Drag Over, debe inicializar $0. Por ejemplo:

```
C_LONGINT($0)
If(Form event=On Drag Over)
   $0:=0
   ...
   If($DataType=Is picture)
      $0:=-1
   End if
   ...
End if
```

Una libertad permitida por el compilador

Las directivas de compilación eliminan toda ambigüedad sobre los tipos de datos. Aunque es necesario cierto rigor, esto no significa que el compilador sea intolerante con cualquier inconsistencia.
Por ejemplo, si asigna un valor real a una variable declarada como de tipo Entero, el compilador no considera que haya un conflicto de tipo y asigna los valores correspondientes de acuerdo a sus directivas. De manera que si escribe:

```
C_LONGINT(vInteger)
vInteger:=2.6
```

El compilador no verá esto como un conflicto de datos que impida la compilación; el compilador simplemente redondeará al valor entero más cercano (3 en lugar de 2.6).

Guía de declaración

Esta sección describe las principales causas de conflictos de tipos en variables, como también las maneras de evitarlos.

Conflictos en variables simples

Los conflictos de tipos simples pueden resumirse en:

- conflictos entre dos usos,
- conflictos entre uso y una directiva de compilación,
- conflictos por declaración implicita,
- conflictos entre dos directivas de compilación.

Conflictos entre dos usos

El conflicto de tipo más simple es aquel en el que un nombre de variable está designado a dos objetos diferentes. Imagine que en una aplicación, usted escribe:

```
Variable:=5
```

y que en otra parte, de la misma aplicación, escribe:

```
Variable:=True
```

Esto genera un conflicto de tipo de datos. El problema puede resolverse renombrando una de las variables.

Conflicto entre el uso y una directiva de compilación

Suponga que escribe en una aplicación:

```
Variable:=5
```

y que en otra parte, en la misma aplicación, escribe:

```
C_BOOLEAN(Variable)
```

Como el compilador primero revisa las directivas, hará la variable de tipo Booleano, pero cuando encuentre:

```
Variable:=5
```

detectará un conflicto de tipo de datos. Puede resolver el problema renombrando la variable o modificando la directiva de compilación.

La utilización de variables de tipos diferentes en una expresión genera inconsistencias. El compilador señala las incompatibilidades. Este es un ejemplo simple:

```
vBool:=True `El compilador deduce que vBoolean es de tipo Booleano
C_LONGINT(<>vInteger) `Declaración de un Entero largo por una directiva de compilación
<>vInteger:=3 `Comando compatible con la directiva de compilación
Var:= <>vInteger+vBool `Operación que contiene variables con tipos de datos incompatibles
```

Conflicto por declaración implícita

Algunas funciones devuelven variables de un tipo muy preciso. La asignación del resultado de una de estas variables a una variable ya declarada de forma diferente provocará un conflicto de tipos si no tiene cuidado.
Por ejemplo, en una aplicación interpretada, puede escribir:

```
IdentNo:=Request("Número de identificación") `IdentNo es de tipo Texto
If(Ok=1)
   IdentNo:=Num(IdentNo) `IdentNo es de tipo Real
   QUERY([Contactos]Id=IdentNo)
End if
```

En este ejemplo, usted genera un conflicto de tipos en la tercera línea. La solución consiste en controlar el comportamiento de la variable. En algunos casos, deberá crear una variable intermedia que utilice un nombre diferente. En otros casos, como este, puede estructurar su método de una forma diferente:

```
IdentNo:=Num(Request("Número de identificación")) `IdentNo es de tipo Real
If(Ok=1)
   QUERY([Contactos]Id=IdentNo)
End if
```

Conflicto entre dos directivas de compilación

Declarar la misma variable por dos directivas de compilación diferentes constituye una redeclaración. Si escribe en la misma base:

```
C_BOOLEAN(Variable)
C_TEXT(Variable)
```

el compilador detecta el conflicto y reporta un error en el archivo de error. Generalmente, puede resolver el problema renombrando una de las variables.

Nota sobre variables locales

Los conflictos de tipo para las variables locales son idénticos a los conflictos de tipo para las variables proceso o interproceso. La única diferencia es que debe haber consistencia sólo en un método específico.
Para las variables proceso e interproceso, los conflictos ocurren a nivel general de la base. Para las variables locales, los conflictos ocurren al nivel del método. Por ejemplo, no puede escribir en el mismo método:

```
$Temp:="Flores"
```

y luego

```
$Temp:=5
```

Sin embargo, puede escribir:

```
$Temp:="Flores"
```

en método M1 y:

```
$Temp:=5
```

en método M2, porque el alcance de las variables locales es el mismo método y no la base completa.

Conflictos en arrays

Los conflictos sobre un array nunca están relacionados con el tamaño del array. En modo compilado como en modo interpretado, los arrays son administrados dinámicamente. El tamaño de un array puede variar a través de los métodos, y no tiene que declarar un tamaño máximo para un array.
Por lo tanto, puede dimensionar un array en cero, añadir o eliminar elementos o borrar el contenido.
Debe seguir las siguientes pautas cuando escribe una base destinada a ser compilada:

- No cambiar los tipos de elementos del array,
- No cambiar el número de dimensiones de un array,
- Para un array Alfa, no cambiar la longitud de las cadenas de caracteres.

Cambio de tipos de los elementos de un array

Si declara un array como un array Entero, debe permanecer como un array Entero en toda la base. Por ejemplo, nunca podrá contener elementos de tipo Booleano.
Si escribe:

```
ARRAY INTEGER(MiArray;5)
ARRAY BOOLEAN(MiArray;5)
```

el compilador no puede identificar el tipo de MiArray.
Simplemente renombre uno de los arrays.

Cambio del número de dimensiones de un array

En una base interpretada, puede cambiar el número de dimensiones de un array. Cuando el compilador establece la tabla de símbolos, se manejan de diferente forma los array de una dimensión que los bidimensionales.
En consecuencia, no puede declarar un array de una dimensión como bidimensional, o viceversa.
Por lo tanto, no puede tener en la misma base:

```
ARRAY INTEGER(MiArray1;10)
ARRAY INTEGER(MiArray1;10;10)
```

Sin embargo, puede escribir en la misma aplicación:

```
ARRAY INTEGER(MiArray1;10)
ARRAY INTEGER(MiArray2;10;10)
```

El número de dimensiones en un array no puede cambiarse en una base. Sin embargo, puede cambiar el tamaño de un array.
Puede redimensionar un array de un array bidimensional y escribir:

```
ARRAY BOOLEAN(MiArray;5)
ARRAY BOOLEAN(MiArray;10)
```

Nota: un array bidimensional, es de hecho, un conjunto de varios arrays de una dimensión. Para mayor información, consulte la sección .

Declaración implícita

Durante la utilización de comandos como **COPY ARRAY**, **LIST TO ARRAY**, **ARRAY TO LIST**, **SELECTION TO ARRAY**, **SELECTION RANGE TO ARRAY**, **ARRAY TO SELECTION**, o **DISTINCT VALUES**, puede cambiar, voluntariamente o no, los tipos de elementos, el número de dimensiones, o en una array alfa, la longitud de la cadena. Usted se encontrará entonces en una de las tres situaciones anteriormente mencionadas.

El compilador genera un mensaje de error; la corrección necesaria es generalmente bastante obvia. Los ejemplos de declaración implícita de arrays están en la sección .

Arrays locales

Si quiere compilar una base que utiliza arrays locales (arrays visibles únicamente por los métodos que los crearon), debe declararlos explícitamente en 4D antes de utilizarlos.
La declaración explícita de un array significa la utilización de un comando de tipo **ARRAY REAL**, **ARRAY INTEGER**, etc.
Por ejemplo, si un método genera un array local de enteros que contiene 10 elementos, usted debe tener la línea siguiente en su método:

```
ARRAY INTEGER($MiArray;10)
```

Declaración de variables creadas en formularios

Las variables creadas en un formulario (como botones, list boxes desplegables, y así sucesivamente) siempre son variables proceso o interproceso.
En una base interpretada, el tipo de estas variables no es importante. Sin embargo, en aplicaciones compiladas, podría tener importancia. No obstante, las reglas son bastante claras:
• Puede declarar variables de formulario utilizando directivas de compilación, o
• El compilador atribuye un tipo por defecto que puede definirse en las Preferencias de compilación (ver el Manual de Diseño).

Variables consideradas por defecto como de tipo numérico

Las siguientes variables de formulario son consideradas por defecto como Reales:

Casilla de selección
Casillas de selección 3D
Botón
Botón inverso
Botón invisible
Botón 3D
Botón imagen
Rejilla de botones
Botón de opción
Botón de opción 3D
Botón imagen de opción
Menú imagen
Menú desplegable jerárquico
Lista jerárquica
Regla
Dial
Termómetro
List box (tipo selección).

Nota: las variables de formulario Regla, Dial y Termómetro siempre son declaradas como Reales, incluso si elige Entero largo como el tipo de botón por defecto en las Preferencias.

Para estas variables, el único conflicto de tipo que puede surgir sería si el nombre de una variable fuera idéntico al de otra ubicada en otra parte de la base. En este caso, renombre la segunda variable.

Variable de área de plug-in

Un área de plug-in siempre es un Entero largo. Nunca podrá haber un conflicto de tipo.
Para un área de plug-in, el único conflicto de tipo que podría surgir es que el nombre de una variable fuera idéntico al de otra ubicada en otra parte de la base. En este caso, renombre la segunda variable.

Variable área 4D Write Pro

Las áreas 4D Write Pro son siempre variables de tipo de Objeto. No puede haber ningún conflicto de escritura, a menos que el mismo nombre de variable se utilice en otra parte de la aplicación.

Variables consideradas por defecto como de Texto

Estas variables son de los siguientes tipos:
Variable no editable,
Variable editable,
Menú/Lista desplegable,
Área de desplazamiento,
Combo box,
Menú Pop-up,
Pestaña,
Área Web,
Columna de list bost (tipo array).

Estas variables están divididas en dos categorías:

- variables simples (variables editables y no editables),
- variables ubicadas en arrays (listas desplegables , menús/listas desplegables, áreas de desplazamiento, menús pop-up, combo boxes y pestañas)

- *variables simples*
  Por defecto, estas variables son de tipo texto. Cuando se utilizan en métodos o en métodos de objeto, se les atribuye el tipo seleccionado por usted. No hay ningún riesgo de conflicto diferente del resultante de asignar el mismo nombre a otra variable.

- *Variables de visualización*
  Algunas variables se utilizan para visualizar los arrays en los formularios. Si los valores por defecto han sido introducidos en el editor de formularios, usted debe declarar explícitamente las variables correspondientes utilizando los comandos de declaración de arrays ([#cmd id="223"/], **ARRAY TEXT**, etc.).

List boxes

Cada list box añade varias variables en los formularios. El tipo por defecto de estas variables depende del tipo de list box:

	List box tipo array	**List box tipo selección**
List box	Array booleano	Número (no se utiliza)
Columna de list box	Array texto	Digitación dinámica
Encabezado	Númerico	Númerico
Pie	Digitación dinámica	Digitación dinámica
Array de control de líneas	Array booleano (Array Entero largo aceptado)	-
Estilos	Array entero largo	Entero largo
Colores de fuente	Array entero largo	Entero largo
Colores de fondo	Array entero largo	Entero largo

Asegúrese de identificar y digitar estas variables y arrays correctamente con el fin de evitar conflictos durante la compilación.

Punteros

Cuando utiliza punteros en su base, toma ventaja de una herramienta poderosa y versátil de 4D. El compilador conserva integralmente los beneficios de los punteros.
Un puntero puede apuntar a variables de tipos de datos diferentes. Asignar diferentes tipos a una variables no crea conflicto. Tenga cuidado de no cambiar el tipo de una variable a la cual un puntero hace referencia.
Este es un ejemplo de este problema:

```
Variable:=5.3
Puntero:=->Variable
Puntero->:=6.4
Puntero->:=False
```

En este caso, su puntero desreferenciado es una variable Real. Al asignarle a esta variable un valor Booleano, se crea un conflicto de tipos.

Si necesita utilizar punteros con diferentes propósitos en el mismo método, asegúrese de que sus punteros estén definidos:

```
Variable:=5.3
Puntero:=->Variable
Puntero->:=6.4
Bool:=True
Puntero:=->Bool
Puntero->:=False
```

Un puntero siempre está definido con relación al objeto al cual se refiere. Por esta razón el compilador no puede detectar conflictos de tipos generados por punteros. En caso de un conflicto, no recibirá un mensaje de error mientras esté en la fase de declaración o de compilación.
Esto no significa que el compilador no tenga manera de detectar conflictos relacionados con punteros. El compilador puede verificar su uso de punteros cuando marca la opción **Control de ejecución** en las Preferencias de compilación (ver Manual de Diseño).

Comandos de plug-ins

Generalidades

Durante la compilación, el compilador analiza las definiciones de los comandos de los plug-ins utilizados en la base, es decir el número y tipo de parámetros de estos comandos. No hay peligro de confusión a nivel de declaración si sus llamadas son consistentes con la declaración del método.

Asegúrese de que sus plug-ins estén instalados en la carpeta **PlugIns**, en una de las ubicaciones autorizadas por 4D: junto al archivo de estructura o junto a la aplicación ejecutable (Windows) / en el paquete de software (Mac OS). Por razones de compatibilidad, aún es posible utilizar la carpeta **Win4DX** o **Mac4DX** junto al archivo de estructura. Para mayor información, consulte la Guía de instalación de 4D.

El compilador no duplica estos archivos, pero los analiza para determinar la declaración adecuada de sus rutinas.
Si sus plug-ins están ubicados en otra parte, el compilador le pedirá ubicarlos durante la declaración, vía una ventana de abrir archivos.

Comandos de plug-in que reciben parámetros implícitos

Ciertos plug-ins, por ejemplo 4D Write, utilizan comandos que llaman implícitamente a comandos 4D.

Tomemos el ejemplo de 4D Write. La sintaxis del comando es:
WR ON EVENT(area;evento;eventoMetodo)

El último parámetro es el nombre del método que usted ha creado en 4D. Este método será llamado por 4D Write cada vez que el evento sea recibido y automáticamente recibe los parámetro siguientes:

Parámetros	Tipo	Descripción
$0	Entero largo	Retorno de función
$1	Entero largo	Área 4D Write
$2	Entero largo	Tecla Mayús.
$3	Entero largo	Tecla Alt (Windows); Tecla Opción (Mac OS)
$4	Entero largo	Tecla Ctrl (Windows), Tecla Comando (Mac OS)
$5	Entero largo	Tipo de evento
$6	Entero largo	Valor que depende del parámetro evento

Para que el compilador tenga en cuenta estos parámetros, debe asegurarse de que hayan sido declarados, bien sea por la directiva de compilación, o por su uso en el método, suficientemente explícito para poder deducir claramente el tipo.

Componentes 4D

4D puede utilizarse para crear y trabajar con componentes. Un componente es un conjunto de objetos 4D representando una o varias funcionalidades agrupadas en un archivo de estructura (llamado base principal), que puede instalarse en diferentes bases (llamadas bases huésped).

Una base huésped ejecutada en modo interpretado puede utilizar indiferentemente componentes interpretados o compilados. Es posible instalar los componentes interpretados y compilados en la misma base huésped. Por el contrario, una base huésped ejecutada en modo compilado no puede utilizar componentes interpretados. En este caso, sólo pueden utilizarse componentes compilados.

Una base huésped interpretada que contiene componentes interpretados puede ser compilada y no llama a los métodos del componente interpretado. Si este no es el caso, aparece una caja de diálogo de alerta cuando la compilación no es posible.

Se puede producir un conflicto de nombre cuando un método de proyecto del componente tiene el mismo nombre que un método de proyecto de la base huésped. En este caso, cuando el código se ejecuta en el contexto de la base huésped, se llama al método de la base huésped. Este principio permite "ocultar" un método del componente con un método personalizado (por ejemplo para obtener una funcionalidad diferente). Cuando el código se ejecuta en el contexto del componente, se llama al método del componente. Este enmascaramiento es señalado como una advertencia durante la compilación de la base huésped.

Si dos componentes comparten métodos con el mismo nombre, se genera un error en el momento de la compilación de la base host.

Para mayor información sobre componentes, consulte el Manual de Diseño.

Manipulación de variables locales $0...$N y paso de parámetros

La manipulación de las variables locales sigue todas las reglas que ya han sido enunciadas. Como las otras variables, sus tipos de datos no pueden alterarse durante la ejecución del método. En esta sección, examinamos dos instancias que pueden conducir a conflictos de tipos:

- Cuando necesita una redeclaración. El uso de punteros ayuda a evitar los conflictos de tipos.
- Cuando necesita direccionar parámetros por indirección.

Utilización de punteros para evitar redeclaraciones.

Una variable no puede ser redeclarada. Sin embargo, es posible utilizar un puntero para referirse a variables de diferentes tipos de datos.
Como ejemplo, considere una función que devuelve el tamaño en memoria de un array de una dimensión. El resultado es un Real, excepto en dos casos; para arrays de tipo texto y de tipo imagen, el tamaño en memoria depende de valores que no pueden expresarse numéricamente (ver la sección **Arrays y memoria**).

En el caso de arrays Texto y arrays Imagen, el resultado se devuelve como una cadena de caracteres. Esta función necesita un parámetro: un puntero al array cuyo tamaño de memoria queremos conocer.

Hay dos métodos para efectuar esta operación:

- Trabajar con variables locales sin preocuparse por su tipo; en tal caso, el método corre sólo en modo interpretado.
- Utilizar punteros y continuar en modo interpretado o compilado.

Función MemSize, en modo interpretado únicamente (Ejemplo para Macintosh)

```
$Tamaño:=Size of array($1->)
$Tipo:=Type($1->)
Case of
   :($Tipo=Real array)
      $0:=8+($Size*10) ` $0 es un Real
   :($Tipo=Integer array)
      $0:=8+($Tamaño*2)
   :($Tipo=LongInt array)
      $0:=8+($Tamaño*4)
   :($Tipo=Date array)
      $0:=8+($Tamaño*6)
   :($Tipo=Text array)
      $0:=String(8+($Tamaño*4))+("+Suma de las longitudes de los textos") ` $0 es un Texto
   :($Tipo=Picture array)
      $0:=String(8+($Tamaño*4))+("+Suma de los tamaños de las imágenes") ` $0 es un Texto
   :($Tipo=Pointer array)
```

```
        $0:=8+($Tamaño*16)
    :($Tipo=Boolean array)
        $0:=8+($Tamaño/8)
End case
```

En el método anterior, el tipo de $0 cambia de acuerdo al valor de $1; por lo tanto, no es compatible con el compilador.

- La función MemSize en modo interpretado y compilado (Ejemplo para Macintosh)
  Acá el método está escrito utilizando punteros:

```
$Tamaño:=Size of array($1->)
$Tipo:=Type($1->)
VarNum:=0
Case of
    :($Tipo=Real array)
        VarNum:=8+($Tamaño*10)  ` VarNum es un Real
    :($Tipo=Integer array)
        VarNum:=8+($Tamaño*2)
    :($Tipo=LongInt array)
        VarNum:=8+($Tamaño*4)
    :($Tipo=Date array)
        VarNum:=8+($Tamaño*6)
    :($Tipo=Text array)
        VarText:=String(8+($Tamaño*4))+("+Suma de longitudes de texto")
    :($Tipo=Picture array)
        VarText:=String(8+($Tamaño*4))+("+Suma de tamaños de imágenes")
    :($Tipo=Pointer array)
        VarNum:=8+($Tamaño*16)
    :($Tipo=Boolean array)
        VarNum:=8+($Tamaño/8)
End case
If(VarNum#0)
    $0:=->VarNum
Else
    $0:=->VarText
End if
```

Estas son las principales diferencias entre las dos funciones:
• En el primer caso, el resultado de la función es la variable que se esperaba,
• En el segundo caso, el resultado de la función es un puntero a esta variable. Usted simplemente desreferencia su resultado.

Indirecciones sobre los parámetros

El compilador administra el poder y la versatilidad de indirección sobre los parámetros. En modo interpretado, 4D le da toda toda la libertad con los números y los tipos de parámetros. Usted mantiene esta libertad en modo compilado, siempre y cuando no introduzca conflictos de tipos y que no utilice más parámetros de los pasados en el método llamado.
Para evitar posibles conflictos, los parámetros direccionados por indirección deben ser del mismo tipo.
Esta indirección es mejor manejada si usted respeta la siguiente convención: si sólo algunos parámetros son direccionados por indirección, deben pasarse después de los otros.
En el método, una dirección por indirección tiene el formato: ${$i}, donde $i es una variable numérica. ${$i} es llamado parámetro genérico.

Como ejemplo, considere una función que añade valores y devuelve la suma con el formato que se pasa como un parámetro. Cada vez que se llama este método, el número de valores a añadir puede variar. Debemos pasar los valores como parámetros al método y el formato en forma de una cadena de caracteres. El número de valores puede variar de llamado en llamado.
Esta función se llama de la forma siguiente:

```
Resultado:=MiSuma("##0.00";125,2;33,5;24)
```

En este caso, el método llamado, obtendrá la cadena "182.70", la cual es la suma de los números, con el formato especificado. Los parámetros de funciones deben pasarse en el orden correcto: primero el formato y luego los valores.

Esta es la función MiSuma:

```
$Sum:=0
For($i;2;Count parameters)
    $Sum:=$Sum+${$i}
End for
$0:=String($Sum;$1)
```

Esta función puede llamarse de varias formas:

```
Resultado:=MySum("##0.00";125,2;33,5;24)
Resultado:=MySum("000";1;18;4;23;17)
```

Al igual que con las otras variables locales, no es necesario declarar parámetros genéricos por directivas de compilación. Si es necesario (en casos de ambigüedad o por optimización), se utiliza la siguiente sintaxis:

```
C_LONGINT($4)
```

Este comando significa que todos los parámetros a partir del cuarto (incluido) estarán direccionados por indirección y serán de tipo Entero largo. $1, $2 y $3 pueden ser de cualquier tipo. Sin embargo, si utiliza $2 por indirección, el tipo utilizado será de tipo genérico. Será de tipo Entero largo, incluso si para usted era, por ejemplo, de tipo Real.

Nota: el compilador utiliza este comando en la fase de declaración. El número en la declaración tiene que ser una constante y no una variable.

Variables reservadas y constantes

Algunas variables y constantes de 4D tiene un tipo y una identidad asignada por el compilador. Por lo tanto, no puede crear una nueva variable, método, función o comando de plug-in comando utilizando cualquiera de los nombres de esta variables o constantes. Puede probar sus valores y utilizarlos como lo hace en modo interpretado.

Variables sistema

Esta es una lista completa de las **Variables sistema** de 4D con sus tipos.

Variable	Tipo
OK	Entero largo
Document	Texto
FldDelimit	Entero largo
RecDelimit	Entero largo
Error	Entero largo
Error method	Texto
Error line	Entero largo
Error formula	Texto
MouseDown	Entero largo
KeyCode	Entero largo
Modifiers	Entero largo
MouseX	Entero largo
MouseY	Entero largo
MouseProc	Entero largo

Variables de informes rápidos

Cuando crea una columna calculada en un informe, 4D crea automáticamente una variable C1 para la primera, C2 para la segunda, C3... y así sucesivamente. Esto se hace de manera transparente.
Si utiliza estas variables en métodos, recuerde que al igual que las otras variables, C1, C2, ... Cn no pueden ser redeclaradas.

Constantes predefinidas 4D

La lista de constantes predefinidas en 4D se encuentra utilizando el **Lista de temas de constantes**. Igualmente puede ver las constantes en Explorador, en modo Diseño.

Detalles de sintaxis

El compilador espera las reglas de sintaxis habituales de los comandos 4D, no necesita modificaciones especiales para bases que van a ser compiladas.

Sin embargo esta sección ofrece algunos recordatorios y detalles específicos:

- Algunos comandos que influyen en el tipo de una variable pueden, si no tiene cuidado, conducir a conflictos.
- Algunos comandos utilizan más de una sintaxis o parámetros, es importante saber cuál es el más apropiado.

Cadenas de caracteres

Character code (carácter)

Para los comandos que operan sobre cadenas, sólo la función **Character code** requiere atención especial. En modo interpretado, puede pasar indiferentemente una cadena no vacía o vacía a esta función.
En modo compilado, no puede pasar una cadena vacía.
Si pasa una cadena vacía y el argumento pasado a **Character code** es una variable, el compilador no podrá detectar un error en la compilación.

Comunicaciones

SEND VARIABLE(variable)
RECEIVE VARIABLE(variable)
Estos dos comandos se utilizan para escribir y recibir variables enviadas al disco. Las variables se pasan como parámetros a estos comandos.

El parámetro que pasa debe ser siempre del mismo tipo. Suponga que quiere enviar una lista de variables a un archivo. Para eliminar el riesgo de cambiar los tipos de datos involuntariamente, recomendamos especificar al principio de la lista el tipo de las variables enviadas. De esta forma, cuando usted recibe estas variables, siempre comenzará por obtener un indicador. Luego, cuando llama a **RECEIVE VARIABLE,** la transferencia se maneja por intermedio de una instrucción **Case of**.

Ejemplo:

```
SET CHANNEL(12;"Archivo")
If(OK=1)
   $Tipo:=Type([Cliente]Total_A)
   SEND VARIABLE($Tipo)
   For($i;1;Records in selection)
      $Enviar_A:=[Cliente]Total_A
      SEND VARIABLE($Enviar_A)
   End for
End if
SET CHANNEL(11)
SET CHANNEL(13;"MiArchivo")
If(OK=1)
   RECEIVE VARIABLE($Tipo)
   Case of
      :($Tipo=ls string var)
         RECEIVE VARIABLE($Cadena)
  `Proceso de la variable recibida
      :($Tipo=ls real)
         RECEIVE VARIABLE($Real)
  `Proceso de la variable recibida
      :($Tipo=ls text)
         RECEIVE VARIABLE($Text)
  `Proceso de la variable recibida
   End case
End if
SET CHANNEL(11)
```

Acceso estructural

Field (puntero campo) o **(tabla número;campo número)**
Table (puntero tabla) o **(tabla número)** o **(puntero campo)**
Estos dos comandos devuelven los resultados de diferentes tipos de datos, de acuerdo a los parámetros pasados:

- Si pasa un puntero a la función **Table**, el resultado devuelto es un número.
- Si pasa un número a la función **Table**, el resultado devuelto es un puntero.

Documentos

Recuerde que las referencias del documento devuelto por las funciones **Open document**, **Append document** y **Create document** son de tipo Hora.

Matemáticas

Mod (valor;divisor)
La expresión "25 modulo 3" puede escribirse de dos formas diferentes en 4d:

```
Variable:=Mod(25;3)
```

o

```
Variable:=25%3
```

El compilador ve una diferencia entre las dos: Mod aplica a todos los tipos numéricos, mientras que el operador % aplica únicamente a Enteros y Enteros largos. Si el operando del operador % excede el rango de los Enteros largos, el resultado devuelto probablemente será errado.

Excepciones

IDLE
ON EVENT CALL (Method{; ProcessName})
ABORT

ON EVENT CALL

El comando **IDLE** se ha añadido al lenguaje 4D para manejar excepciones. Este comando debe utilizarse cuando utilice el comando **ON EVENT CALL**.

Este comando puede definirse como una directiva de gestión de eventos.
Sólo el kernel (núcleo) de 4D puede detectar un evento sistema (clic del ratón, actividad del teclado, etc.). En la mayoría de los casos, las llamadas al kernel son iniciadas por el mismo código compilado, de una manera transparente para el usuario.

Por otra parte, cuando 4D está esperando pasivamente por un evento, por ejemplo en un bucle de espera, es evidente que no habrá ninguna llamada.

Ejemplo bajo Windows

```
  `Método de proyecto ClicRaton
If(MouseDown=1)
   <>vPrueba:=True
   ALERT("Alguien hizo clic en el ratón")
End if

  `Método Espera
<>vPrueba:=False
ON EVENT CALL("ClicRaton")
While(<>vPrueba=False)
  `Bucle de atención del evento
End while
ON EVENT CALL("")
```

En este caso, se añade el comando **IDLE** de la forma siguiente:

```
  `Método Espera
<>vPrueba:=False
ON EVENT CALL("ClicRaton")
While(<>vPrueba=False)
   IDLE
  `Llamar al Kernel para detectar un evento
End while
ON EVENT CALL("")
```

ABORT

Utilice este comando sólo en métodos de proyecto de intercepción de errores. Este comando funciona exáctamente como en 4D, excepto en un método que ha sido llamado por uno de los siguientes comandos: **EXECUTE FORMULA**, **APPLY TO SELECTION** y **APPLY TO SUBSELECTION**. Trate de evitar esta situación.

Arrays

Siete comandos de 4D son utilizados por el compilador para determinar el tipo de un array:

COPY ARRAY(fuente;destino)
SELECTION TO ARRAY(campo;Array)

ARRAY TO SELECTION(Array;campo)
SELECTION RANGE TO ARRAY(inicio;fin;campo;Array)
LIST TO ARRAY(lista;Array{; itemRefs})
ARRAY TO LIST(Array;lista{; itemRefs})
DISTINCT VALUES(campo;Array)

COPY ARRAY

El comando **COPY ARRAY** acepta dos parámetros de tipo Array. Si uno de los parámetros de array no ha sido declarado, el compilador determina el tipo del array no declarado según el tipo del declarado.
Esta deducción se realiza en los dos casos siguientes:

- El array declarado es el primer parámetro. El compilador asigna el tipo del primer array al segundo array.
- El array declarado es el segundo parámetro. En este caso, el compilador asigna el tipo de dato del segundo array al primer array.

Como el compilador es estricto con los tipos, **COPY ARRAY** sólo puede hacerse desde un array de un cierto tipo a un array del mismo tipo.
Por lo tanto, si quiere copiar un array de elementos cuyos tipos son similares, es decir, Enteros, Enteros largos y Reales, o Texto y Alfas, o Alfas de diferentes longitudes, tiene que copiar los elementos uno por uno.

Imagine que quiere copiar elementos de un Array Entero a un Array Real. Puede proceder de la siguiente forma:

```
$Tamaño:=Size of array(ArrInt)
ARRAY REAL(ArrReal;$Tamaño)
  `Establecer el mismo tamaño para el array Real que para el array Entero
For($i;1;$Size)
    ArrReal{$i}:=ArrInt{$i}
  `Copiar cada elemento
End for
```

Recuerde que no puede cambiar el número de dimensiones de un array durante el proceso. Si copia un array de una dimensión en un array de bidimensional, el compilador genera un mensaje de error.

SELECTION TO ARRAY, ARRAY TO SELECTION, DISTINCT VALUES, SELECTION RANGE TO ARRAY

De la misma forma para 4D en modo interpretado, estos cuatro comandos no necesitan la declaración de arrays. Los arrays no declarados reciben el mismo tipo que el campo especificado en el comando.
Si escribe:

```
SELECTION TO ARRAY([MiTabla]CampoEntero;MiArray)
```

el tipo de datos de MiArray sería Entero de una dimensión. (asumiendo que CampoEntero es un campo entero).

Si el array ha sido declarado, asegúrese de que el campo sea del mismo tipo. Aunque Entero, Entero largo y Real son tipos similares, no son equivalentes.
Por otra parte, en el caso de tipos Texto y Alfa, tiene un poco más de latitud. Por defecto, si un array no fue declarado previamente y usted aplica un comando que incluye un campo de tipo Alfa como parámetro, se atribuye el tipo Texto al array. Si el array fue declarado previamente como de tipo Alfa o de tipo Texto, estos comandos seguirán sus directivas.

Lo mismo ocurre en el caso de los campos de tipo Texto--sus directivas tienen prioridad.
Recuerde que los comandos **SELECTION TO ARRAY**, **SELECTION RANGE TO ARRAY**, **ARRAY TO SELECTION** y **DISTINCT VALUES** sólo pueden utilizarse con arrays de una dimensión.

El comando **SELECTION TO ARRAY** también tiene una segunda sintaxis:
SELECTION TO ARRAY(tabla;Array).
En este caso, la variable MiArray será de Array de Enteros largos. El comando **SELECTION RANGE TO ARRAY** funciona de la misma forma.

LIST TO ARRAY, ARRAY TO LIST

Los comandos **LIST TO ARRAY** y **ARRAY TO LIST** conciernen sólo a dos tipos de arrays:

- los arrays Alfa de una dimensión y
- los arrays Texto de una dimesnión.
  Estos comandos no necesitan la declaración del array que se pasa como parámetro. Por defecto, un array no declarado recibirá el tipo Texto. Si el Array fue declarado previamente como de tipo Alfa o Texto, estos comandos seguirán sus directivas.

Utilización de punteros en comandos relacionados con arrays

El compilador no puede detectar un conflicto de tipo si usted utiliza un puntero sin referencia como parámetro de un comando de declaración de un array. Si escribe:

```
SELECTION TO ARRAY([Tabla]Campo;Puntero->)
```

donde Puntero-> representa un array, el compilador no puede verificar que el tipo del campo y el del array son idénticos. Debe prevenir tales conflictos; debe declarar el array referenciado por el puntero.

El compilador emite una advertencia cuando encuentra una rutina de declaración de array en la cual uno de los parámetros es un puntero. Estos mensajes pueden ser útiles en la detección de este tipo de conflicto.

Arrays locales

Si su base utiliza arrays locales (arrays reconocidos únicamente en el método en el que fueron creados), es necesario declararlos explícitamente en 4D antes de utilizarlos.

Para declarar un array local, utilice uno de los comandos de array tales como **ARRAY REAL**, **ARRAY INTEGER**, etc.
Por ejemplo, si un método crea un array local de enteros que contiene 10 elementos, debe declarar el array antes de utilizarlo. Utilice el comando:

```
ARRAY INTEGER($MiArray;10)
```

Lenguaje

Get pointer(varNombre)
Type (objeto)
EXECUTE FORMULA(instrucción)
TRACE
NO TRACE

Get pointer

Get pointer es una función que devuelve un puntero al parámetro que usted le pasó. Suponga que quiere inicializar un array de punteros. Cada elemento en ese array apunta a una variable dada. Suponga que hay doce variables llamadas V1, V2, …V12. Puede escribir:

```
ARRAY POINTER(Arr;12)
Arr{1}:=->V1
Arr{2}:=->V2

Arr{12}:=->V12
```

También puede escribir:

```
ARRAY POINTER(Arr;12)
For($i;1;12)
   Arr{$i}:=Get pointer("V"+String($i))
End for
```

Al final de esta operación, puede obtener un array de punteros donde cada elemento apunta a una variable Vi.
Estas dos secuencias pueden compilarse. Sin embargo, si las variables V1 a V12 no se utilizan explícitamente en otra parte de la base, el compilador no puede darles un tipo. Por lo tanto, deben ser utilizadas o declaradas explícitamente en otra parte.
Esta declaración explícita puede realizarse de dos formas:

- Declarando V1, V2, …V12 a través de una directiva de compilación:

```
C_LONGINT(V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12)
```

- Asignando estas variables en un método:

```
V1:=0
V2:=0

V12:=0
```

Type (objeto)

Como cada variable en una base compilada tiene un solo tipo, esta función puede parecer no muy útil. Sin embargo, puede ser útil cuando trabaja con punteros. Por ejemplo, puede ser necesario conocer el tipo de la variable a la cual el puntero hace referencia; debido a la flexibilidad de los punteros, no siempre es posible saber a que objeto apuntan.

EXECUTE FORMULA

Este comando ofrece ventajas en modo interpretado que no ofrece en modo compilado.
En modo compilado, un nombre de método pasado como parámetro a este comando será interpretado. Por lo tanto, usted no se beneficia de las ventajas ofrecidas por el compilador y la sintaxis de su parámetro no podrá verificarse.
Además, usted no puede pasar variables locales como parámetros.
Puede reemplazar **EXECUTE FORMULA** por una serie de instrucciones. Estos son dos ejemplos.
Dada la siguiente secuencia:

```
i:=FormFunc
EXECUTE FORMULA("FORM SET INPUT(Form"+String(i)+")")
```

Puede reemplazarla por:

```
i:=FormFunc
VarForm:="Form"+String(i)
FORM SET INPUT(VarForm)
```

Miremos este otro ejemplo:

```
$Num:=SelPrinter
EXECUTE FORMULA("Print"+$Num)
```

Acá, **EXECUTE FORMULA** puede reemplazarse por **Case of**:

```
Case of
   :($Num=1)
      Print1
   :($Num=2)
      Print2
   :($Num=3)
      Print3
End case
```

El comando **EXECUTE FORMULA** puede reemplazarse siempre. Como el método a ejecutar se elije de la lista de los métodos de proyecto de la base o de los comandos de 4D, hay un número finito de métodos. Por lo tanto, siempre es posible reemplazar el comando **EXECUTE FORMULA** por Case of o por otro comando. Además, su código se ejecutará más rápido.

TRACE, NO TRACE

Estos dos comandos se utilizan en el proceso de depuración. No tienen ningún objetivo en una base compilada. Sin embargo, puede manterlos en sus métodos, simplemente serán ignorados por el compilador.

Variables

Undefined(variable)
SAVE VARIABLES(documento;variable1{; variable2...})
LOAD VARIABLES(documento;variable1{; variable2...})
CLEAR VARIABLE(variable)

Undefined

Teniendo en cuenta el proceso de declaración efectuado por el compilador, una variable no puede en ningún momento indefinida en modo compilado. De hecho, todas las variables han sido definidas en el momento en que termina la compilación. La función **Undefined** por lo tanto siempre devuelve **False**, sin importar el parámetro que se pase.

Nota: para saber si su aplicación está corriendo en modo compilado, llame el comando **Compiled application**.

SAVE VARIABLES, LOAD VARIABLES

En modo interpretado, puede verificar que el documento existe probando si una de las variables está indefinida después de la ejecución de **LOAD VARIABLES**. Esto no es posible en bases compiladas, ya que la función **Undefined** siempre devuelve **False**.

Esta prueba puede realizarse en modo interpretado o compilado:
1. Inicialice las variables que va a recibir en un valor que no sea un valor legal para cualquiera de las variables.
2. Compare una de las variables recibidas con el valor de inicialización después de **LOAD VARIABLES**.

El método se puede escribir de este modo:

```
Var1:="xxxxxx"
  `"xxxxxx" es un valor que no puede ser devuelto por LOAD VARIABLES
Var2:="xxxxxx"
Var3:="xxxxxx"
Var4:="xxxxxx"
LOAD VARIABLES("Documento";Var1;Var2;Var3;Var4)
If(Var1="xxxxxx")
  `Documento no encontrado

Else
  `Documento encontrado

End if
```

CLEAR VARIABLE

Esta rutina utiliza dos sintaxis diferentes en modo interpretado:
CLEAR VARIABLE(variable)
CLEAR VARIABLE("a")
En modo compilado, la primera sintaxis de **CLEAR VARIABLE(variable)** reinicializa la variable (puesta en cero para un numérico; cadena vacía para un cadena de caracteres o un texto, etc.), ya que ninguna variable puede estar indefinida en modo compilado.
Por lo tanto, **CLEAR VARIABLE** no libera memoria en modo compilado, excepto en cuatro casos: las variable de tipo Texto, Imagen, BLOB y Arrays.
Para un Array, **CLEAR VARIABLE** tiene el mismo efecto que una nueva declaración del array donde el tamaño se establece como cero.

Para un array MiArray cuyos elementos son de tipo *Entero*, **CLEAR VARIABLE(MiArray)** tiene el mismo efecto de una de las expresiones siguientes:

```
ARRAY INTEGER(MiArray;0)
  `si es un array de una dimensión
```

```
ARRAY INTEGER(MiArray;0;0)
  `si es un array de dos dimensiones
```

La segunda sintaxis, **CLEAR VARIABLE("a")**, no es compatible con el compilador, ya que el compilador accede a las variables por dirección, no por nombre.

Punteros con algunos comandos

Punteros con algunos comandos

Los siguientes comandos tienen una característica en común: aceptan un primer parámetro opcional [Tabla] y el segundo parámetro puede ser un puntero.

ADD TO SET	**LOAD SET**
APPLY TO SELECTION	**LOCKED BY**
COPY NAMED SELECTION	**ORDER BY**
CREATE EMPTY SET	**ORDER BY FORMULA**
CREATE SET	**FORM SET OUTPUT**
CUT NAMED SELECTION	**PAGE SETUP**
DIALOG	**Print form**
EXPORT DIF	**PRINT LABEL**
EXPORT SYLK	**QR REPORT**
EXPORT TEXT	**QUERY**
GOTO RECORD	**QUERY BY FORMULA**
GOTO SELECTED RECORD	**QUERY SELECTION**
_o_GRAPH TABLE	**QUERY SELECTION BY FORMULA**
IMPORT DIF	**REDUCE SELECTION**
IMPORT SYLK	**RELATE MANY**
IMPORT TEXT	**REMOVE FROM SET**
FORM SET INPUT	

En modo compilado, es fácil devolver el parámetro opcional [Tabla]. Sin embargo, cuando el primer parámetro pasado a uno de estos comandos es un puntero, el compilador no sabe a que puntero está haciendo referencia; el compilador lo trata como un puntero de tabla.

Tomemos el caso del comando **QUERY** cuya sintaxis es la siguiente:
QUERY({tabla{;formula{;*}}})

El primer elemento del parámetro *formula* debe ser un campo.

Si escribe:

```
QUERY(PtrField->=True)
```

el compilador buscará un símbolo que represente un campo en el segundo elemento. Cuando encuentra el signo "=", emitirá un mensaje de error, al no poder identificar el comando con una expresión que sepa tratar.

Por otra parte, si escribe:

```
QUERY(PtrTabla->;PtrCampo->=True)
```

o

```
QUERY([Tabla];PtrCampo->=True)
```

evitará cualquier ambigüedad posible.

Uso directo de los comandos que devuelven punteros

Cuando se utilizan punteros, hay una particularidad relativa a los comandos en el que el primer parámetro [unaTabla] y el segundo parámetro son opcionales. En este contexto, por razones internas, el compilador no permite un comando que devuelva un puntero (por ejemplo **Current form table**) para ser pasado directamente como un parámetro (se genera un error).

Este es el caso, por ejemplo, del comando **FORM SCREENSHOT**. El código siguiente funciona en modo interpretado, pero es rechazado durante la compilación:

```
 //dispara un error de compilación
FORM SCREENSHOT(Current form table->;$formName;$myPict)
```

En este caso, sólo puede usar una variable intermedia para que este código sea validado por el compilador:

```
 //código equivalente compilable.
C_POINTER($ptr)
$ptr:=Current form table
FORM SCREENSHOT($ptr->;$formName;$myPict)
```

Constantes

Si crea sus propios recursos 4DK# (constantes), asegúrese de que los numéricos sean declarados como de tipo Entero largo (L) o Reales (R) y las cadenas de caracteres como Cadenas (S). Cualquier otro tipo generará una advertencia.

Consejos de optimización

Es difícil establecer un método definitivo para "programar bien", pero queremos hacer énfasis en las ventajas de estructurar bien los programas. La capacidad de programar estructuradamente en 4D puede ser de gran ayuda.

La compilación de una base bien estructurada puede obtener mejores resultados que en una base mal estructurada. Por ejemplo, si escribe un método genérico para manejar n métodos de objeto, obtendrá resultados de más calidad en modo interpretado y en compilado que en una situación donde n métodos de objeto componen n veces el mismo conjunto de instrucciones.

En otras palabras, la calidad de la programación tiene un impacto en la calidad del código compilado.
Con práctica, puede mejorar progresivamente su código 4D. El uso frecuente del compilador le da una retroalimentación correctiva que le permite alcanzar instintivamente la solución más eficiente.

Entre tanto, le podemos ofrecer algunos consejos y trucos para que ahorre tiempo realizando tareas simples y recurrentes.

Uso de comentarios en el código

Algunas técnicas de programación pueden hacer que su código no sea fácil de leer tanto para usted como para otra persona en el futuro. Por esta razón, le aconsejamos comentar detalladamente sus métodos. De hecho, mientras que los comentarios excesivos tienden a hacer lenta una base interpretada, no tienen ninguna influencia en el tiempo de ejecución de una base compilada.

Uso de directivas de compilación para optimizar el código

Las directivas de compilación pueden ayudarle a acelerar su código considerablemente. Cuando declara variables con base en su uso, el compilador utiliza el tipo de datos con el mayor alcance posible para no penalizarlo. Por ejemplo, si no declara la variable definida por la instrucción: **Var:= 5**, el compilador le dará el tipo Real, incluso si se puede declarar como Entero.

Reales

Por defecto, el compilador da el tipo Real a las variables numéricas no declaradas por directivas de compilación y si las Propiedades de la base no indican otra cosa. Los cálculos con Reales son más lentos que con Enteros largos. Si sabe que una variable numérica siempre será un entero, es conveniente declararla con las directivas de compilación **C_LONGINT**.

Por ejemplo, es una buena práctica declarar sus contadores de bucles como Enteros.

Algunas funciones estándar de 4D devuelven Enteros (ejemplo: **Character code**, **Int**...). Si usted asigna el resultado de una estas funciones a una variable no declarada de su base, el compilador le asigna el tipo Real en lugar del tipo Entero. Recuerde declarar estas variables con directivas de compilación siempre que esté seguro de que no se utilizarán en un contexto diferente.

Este es un ejemplo simple de una función que devuelve un valor aleatorio dentro de un rango dado:

```
$0:=Mod(Random;($2-$1+1))+$1
```

Siempre devolverá un entero. Escrito de esta manera, el compilador le dará a $0 el tipo Real en lugar de Entero o Entero largo. Es mejor incluir una directiva de compilación en el método:

```
C_LONGINT($0)
$0:=Mod(Random;($2-$1+1))+$1El parámetro devuelto por el método ocupará menos espacio en memoria y será much.

Este es otro ejemplo. Imagine que declara dos variables como Entero largo:

C_LONGINT($var1;$var2)
```

y una tercera variable no declarada recibe la suma de las otras dos variables:$var3:=$var1+$var2.

El compilador declarará la tercera variable, $var3, como Real. Usted tendrá que declararla explícitamente como Entero largo si quiere que el resultado sea Entero largo.

Nota: sea cuidadoso con el modo de cálculo en 4D. En modo compilado, no es el tipo de la variable que recibe el cálculo el que determina el modo de cálculo, sino el tipo de los operandos.

• En el siguiente ejemplo, el cálculo se efectúa sobre enteros largos:

```
C_REAL($var3)
C_LONGINT($var1;$var2)
$var1:=2147483647
$var2:=1
$var3:=$var1+$var2
```

$var3 es igual a –2147483648 tanto en modo compilado como interpretado.

- Sin embargo, en este ejemplo:

```
C_REAL($var3)
C_LONGINT($var1)
```

```
$var1:=2147483647
$var3:=$var1+1
```

Por razones de optimización, el compilador considera el valor 1 como un entero. En modo compilado, $var3 es igual a –2147483648 porque el cálculo se efectúa sobre Enteros largos. En modo interpretado, $var3 es igual a 2147483648 porque el cálculo se efectúa sobre Reales.

Los botones son un caso específico de Real que puede declararse como Entero largo.

Cadenas de caracteres

Si quiere probar el valor de un carácter, es más rápido hacer la comparación con su valor **Character code** en vez de con el carácter mismo. La rutina estándar de comparación de caracteres tiene en cuenta todas las variaciones del carácter, tales como las acentuaciones.

Diferentes observaciones

Arrays de dos dimensiones

El procesamiento de arrays de dos dimensiones se maneja mucho mejor si la segunda dimensión es más larga que la primera.

Por ejemplo, un array declarado como:ARRAY INTEGER(Array;5;1000)

será manejado mejor que un array declarado como:

```
ARRAY INTEGER(Array;1000;5)
```

Campos

Cuando necesite efectuar varios cálculos sobre un campo, puede mejorar el desempeño almacenando el valor del campo en una variable y realizando sus cálculos sobre la variable en lugar de sobre el campo. Considere el siguiente método:

```
Case of
   :([Cliente]Dest="Nueva York")
      Transporte:="Mensajero"
   :([Cliente]Dest="Puerto Rico")
      Transporte:="Correo aéreo"
   :([Cliente]Dest="Exterior")
      Transporte:="Servicio de correo expreso"
   Else
      Transporte:="Servicio de correo regular"
End case
```

Este método tomará más tiempo en ejecutarse que si se escribe de esta forma:

```
$Dest:=[Cliente]Dest
Case of
   :($Dest="Nueva York")
      Transporte:="Mensajero"
   :($Dest="Puerto Rico")
      Transporte:="Correo aéreo"
   :($Dest="Exterior")
      Transporte:="Servicio de correo expreso"
   Else
      Transporte:="Servicio de correo regular"
End case
```

Punteros

Como en el caso de los campos, es más rápido trabajar con variables que con punteros sin referencia. Cuando necesite realizar varios cálculos sobre una variable referenciada por un puntero, puede ahorrar tiempo almacenando el puntero sin referencia en una variable.

Por ejemplo, suponga que utiliza un puntero, MiPtr, para hacer referencia a un campo o a una variable. Usted quiere realizar un conjunto de pruebas sobre el valor referenciado por MiPtr. Puede escribir:

```
Case of
   :(MiPtr->=1)
      Secuencia 1
   :(MiPtr->=2)
      Secuencia 2

End case
```

El conjunto de pruebas se realizaría más rápido si se escribiera:

```
Temp:=MiPtr->
Case of
   :(Temp=1)
```

```
        Secuencia 1
    :(Temp=2)
        Secuencia 2

End case
```

Variables locales

Utilice variables locales, siempre que sea posible, para estructurar su código. Utilizar variables locales tiene las siguientes ventajas:

- Las variables locales ocupan menos espacio cuando se utilizan en una base. Se crean cuando se entra en el método donde se utilizan y se destruyen cuando el método termina su ejecución.
- El código generado se optimiza para variables locales, especialmente para las de tipo Entero largo. Esto es útil para contadores en bucles.

Mensajes de error

Esta sección describe los diferentes mensajes generados por el compilador. Estos mensajes son de diferentes tipos:

- advertencias, que ayudan a evitar trampas comunes;
- errores, que usted debe corregir;
- mensajes de control de ejecución, generados dentro de 4D.

Advertencias

Estos mensajes se generan durante el proceso de compilación. Cada mensaje está acompañado de un ejemplo del problema y cuando es necesario, de una explicación adicional.

Uso de puntero(s) como parámetro(s) de COPY ARRAY

```
COPY ARRAY(Puntero->;Array)
```

Uso de puntero(s) como parámetro(s) de SELECTION TO ARRAY

```
SELECTION TO ARRAY(Puntero->;MiArray)
SELECTION TO ARRAY([MiTabla]MiCampo;Puntero->)
```

Uso de puntero(s) como parámetro(s) de ARRAY TO SELECTION

```
ARRAY TO SELECTION(Puntero->;[MiTabla]MiCampo)
```

Uso de puntero(s) como parámetro(s) de LIST TO ARRAY

```
LIST TO ARRAY(Lista;Puntero->)
```

Uso de puntero(s) como parámetro(s) de ARRAY TO LIST

```
ARRAY TO LIST(Puntero->;Lista)
```

Uso de puntero en una declaración de array

```
ARRAY REAL(Puntero->;5)
```

El comando **ARRAY REAL(Array;Puntero->)** no genera esta advertencia. El valor de la dimensión de un array no tiene ninguna influencia sobre su tipo. En este ejemplo, el array al que hace referencia el puntero debe haber sido definido en otra parte.

Uso de puntero(s) como parámetro(s) de DISTINCT VALUES

```
DISTINCT VALUES(Puntero->;Array)
```

Utilización de la función Undefined.

```
If(Undefined(Variable))
```

La función **Undefined** siempre devuelve *FALSE* en una base compilada.
Este método está protegido por una contraseña.
Un botón con acción automática no tiene nombre en la página X del formulario MiForm.
Todos sus botones deben tener nombres para evitar conflictos.
Asume que el puntero apunta a una expresión alfanumérica.

```
Pointer->≤2≥:="a"
```

Asume que la cadena index es numérica.

```
String≤Pointer->≥:="a"
```

Asume que el índice del array es de tipo real.

```
ALERT(MyArray{Pointer->})
```

Falta un parámetro en la llamada al comando del plug-in.

```
WR SET FONT(Area)
```

Nota: es posible desactivar y activar individualmente las advertencias con las siguientes etiquetas:
//%W-número_de_advertencia para desactivar una advertencia
//%W+número_de_advertencia para activar una advertencia
Estas activaciones y desactivaciones son efectivas para todo el código analizado en el plan de compilación. Si quiere desactivar o activar una advertencia de manera global, inserte la etiqueta apropiada en un método llamado "Compiler_xxx" ya que estos métodos son analizados primero por el compilador. Por ejemplo, para desactivar la advertencia "Puntero en COPY ARRAY", puede insertar la etiqueta "//%W-518.1" en la ubicación deseada.

Errores

Estos mensajes se generan durante el proceso de compilación. Le corresponde a usted corregir estos errores para que el compilador pueda generar una base compilada. Cada mensaje está acompañado de un ejemplo del problema y cuando es necesario, de una explicación adicional.
Declaración
El tipo de la variable no es compatible con el operador. No es posible hacer una instrucción con estos tipos.

```
MyBoolean:=True
MyReal:=MyBoolean
```

Cambiando el número de dimensiones de un array.

```
ARRAY TEXT(MiArray;5;5)
ARRAY TEXT(MiArray;5)
```

Conflicto de tipo sobre la variable MiArray en el formulario.

```
ARRAY INTEGER(MiArray)
```

Declarando un array sin dimensiones.

```
ARRAY INTEGER(MiArray)
```

Variable esperada

```
COPY ARRAY(MiArray;"")
```

El tipo de la variable es desconocido. Esta variable se utiliza en el método M1.
No se puede determinar el tipo de variable. Es necesaria una directiva de compilación.
Tipo de constante incorrecto

```
OK:="El clima es agradable"
```

El método M1 es desconocido.
La línea contiene una llamada a un método que no existe o no existe más.
Incorrecto uso de un campo.

```
MiFecha:=Add to date(CampoBooleano;1;1;1)
```

La variable Variable no es un método.

```
Variable(1)
```

La variable Variable no es un array.

```
Variable{5}:=12
```

El resultado de la función no es compatible con la expresión.

```
Texto:="Número"+Num(i)
```

El tipo de la variable no es compatible con el operador

```
Entero:=MiFecha*Texto
```

No se puede efectuar una asignación con estos tipos.

```
$i:="3"
$($i):=5
```

El índice del array no es numérico.

```
EntArray{"3"}:=4
```

No se puede efectuar una asignación con estos tipos

```
C_TEXT(Variable)
COPY ARRAY(TextoArray;Variable)
```

No se puede efectuar una asignación con estos tipos

```
Variable:=Num(Variable)
```

No se puede efectuar una asignación con estos tipos

```
Variable:=MiBooleano
```

Cambiando el array EntARRAY de tipo Entero a una variable de tipo Texto

```
ARRAY TEXT(EntArray;12)
```

si EntArray fue declarado en otra parte como un array de enteros.
Intentando desreferenciar una variable que no es de tipo puntero.

```
Variable->:=5
```

si Variable no es de tipo puntero.
Redeclaración de la variable Var1 de tipo texto a tipo numérico.

```
Var1:=3.5
```

Incorrecto uso de un campo.

```
Variable:=[MiTabla]MiCampo
```

[MiTabla]MiCampo es un campo tipo Fecha. Variable es de tipo Numérico.

Sintaxis

El resultado de la función no es un puntero.

```
Variable:=Num("El clima es agradable")->
```

No es posible desreferenciar esta función.
Error de sintaxis.

```
If(Booleano)
End for
```

Demasiados corchetes de apertura ({) .
La línea contiene más corchetes de apertura que de cierre.
Demasiados corchetes de cierre (})..
La línea contiene más corchetes de cierre que de apertura.
Falta un paréntesis de cierre).
La línea contiene más paréntesis de apertura que de cierre.
Falta un paréntesis de apertura (.
La línea contiene más paréntesis de cierre que de apertura.
Falta un campo.

```
If(Modified(Variable))
```

Falta un paréntesis de cierre).

```
C_INTEGER($
```

Falta una variable.

```
C_INTEGER([MiTabla]MiCampo)
```

Tipo de constante incorrecto : Alfanumérico

```
C_INTEGER(${"3"})
```

Falta un punto y coma (;).

```
COPY ARRAY(Array1 Array2)
```

Mac OS

- Demasiados caracteres de apertura para los índices de la cadena.

```
MiCadena≤3:="a"
```

Demasiados caracteres de cierre para los índices de la cadena.

```
MiCadena3≥:="a"
```

Windows

Demasiados caracteres de apertura para los índices de la cadena.

```
MiCadena[[3:="a"
```

Demasiados caracteres de cierre para los índices de la cadena.

```
MiCadena 3]]:="a"
```

No se esperaba una subtabla.

```
ARRAY TO SELECTION(Array;Subtabla)
```

El parámetro de una clausula IF debe ser un booleano.

```
If(Puntero)
```

La expresión es muy compleja.
Divida su instrucción en varias instrucciones más cortas.

El método es muy complejo.
Demasiadas estructuras Case of...End case y/o If...End if.

Campo desconocido.
Su método, probablemente copiado de otra base, contiene •???• en lugar de un nombre de campo.

Tabla desconocida.
Su método, posiblemente copiado de otra base, contiene •???• en lugar de un nombre de tabla.

Puntero a una expresión incorrecta.

```
Puntero:=->Variable+3
```

Uso incorrecto de índice de cadena.

```
MiReal≤3≥o MiReal[[3]]
```

Utilización incorrecta de la Variable VARIABLE de Tipo Texto como una variable de tipo Real

```
MiCadena≤Variable≥o MiCadena[[Variable]]
```

donde Variable no es una variable numérica.

Parámetros

El resultado de esta función no es compatible con la expresión.

```
MiMetodo(Num(MiCadena))
```

si MiMetodo espera una expresión de tipo Booleano.

Se han pasado demasiados parámetros a este método.

```
DEFAULT TABLE(Tabla;Form)
```

El resultado de esta función no es compatible con la expresión.

```
MiMetodo(3+2)
```

si MiMetodo espera una expresión Booleana.
El resultado de esta función no es compatible con la expresión.

```
C_INTEGER($0)
$0:=False
```

El resultado de esta función no es compatible con la expresión.

```
C_INTEGER(${3})
For($i;3;5)
   ${$i}:=String($i)
End for
```

Este comando no requiere ningún parámetro

```
SHOW TOOL BAR(MiVar)
```

Este comando requiere al menos un parámetro.

```
DEFAULT TABLE
```

MiCadena no puede pasarse como parámetro a este método.

```
MiMetodo(MiCadena)
```

si MiMetodo está esperando un parámetro Booleano.
El tipo del parámetro $1 es diferente en el llamado y en el método llamado.

```
Calcular("3+2")
```

con la directiva **C_INTEGER($1)** en Calcular.
Uno de los parámetros en COPY ARRAY es una variable.

```
COPY ARRAY(Variable;Array)
```

Redeclaración de la variable $1 de tipo Real a tipo Texto.

```
$1:=String($1)
```

Un array no puede ser un parámetro.
ReInit(MiArray)Para pasar un array a un método, necesita para un puntero al array.

Operadores

El tipo de variable no es compatible con el operador.

```
Bool2:=Bool1+True
```

La suma no puede realizarse sobre campos Booleanos.
No se esperaba el operador >.

```
QUERY([MiTabla];[MiTabla]MiCampo=0;>)
```

No se pueden compara dos variables de estos tipos

```
If(Numero=Imagen2)
```

No se puede transformar en negativo este tipo de variable.

```
Boolean:=-False
```

Comandos de plug-ins

El comando de plug-in PExt no parece estar definido correctamente.
No se pasaron suficientes parámetros a este comando de plug-in.
Se pasaron demasiados parámetros a este comando de plug-in.
El comando Variable del plug-in no parece estar correctamente definido.

Errores generales

Dos métodos tienen el mismo nombre : Nombre.
Para compilar su base, todos los métodos de proyecto deben tener nombres diferentes.
Error interno # xx.
Si aparece este mensaje, llame al Soporte técnico de 4D e informe el número del error.

La variable Variable no pudo ser declarada. Esta variable se utiliza en el método M1.
El tipo de la variable no puede ser determinado. Es necesaria una directiva de compilación.

El método original está dañado.
El método está dañado en la estructura original. Bórrelo o reemplácelo.

Comando 4D desconocido.
El método está dañado.

Redeclaración de la variable Variable en el formulario Formulario.
Este mensaje aparece si usted da, por ejemplo, el nombre OK a una variable de tipo gráfico en un formulario.

El nombre de la función es también el nombre de una variable en el formulario.
Renombre la función o la variable.

Un método y una variable tienen el mismo nombre : Nombre.
Renombre el método o la variable.

Un comando de plug-in y una variable tienen el mismo nombre: Nombre.
Renombre el comando de plug-in comando o la variable.

No se puede llamar un comando que no es hilo seguro de un método declarado como hilo seguro.Modifique el método para que sea hilo seguro (no utilizar cualquier comando que no sea hilo seguro) o cambiar la propiedad de declaración del método para iniciar un proceso cooperativo

No puede llamar al método 'MethodName' que no es hilo seguro desde un método declarado hilo seguro.
Modificar el método para que sea hilo seguro o cambiar la propiedad de declaración del método con el fin de iniciar un proceso cooperativo.

Mensajes de control de ejecución

Estos mensajes son generados por 4D durante la ejecución de la base compilada y son visualizados en una ventana de error específica.

El resultado está por fuera del rango.
Si MiArray es un array de 5 elementos, este mensaje aparece si usted trata de acceder al elemento 17 en el array.

División por cero.

```
Var1:=0
Var2:=2
Var3:=Var2/Var1
```

Acceso a un parámetro que no existe.
Utilización de la variable local $4 cuando sólo tres parámetros han sido pasados al método actual.

El puntero no está correctamente inicializado.

```
MiPuntero->:=5
```

si MiPuntero no ha sido inicializado aún.

La cadena de destino es más pequeña que la fuente.

```
C_STRING(MiCadena1;5)
C_STRING(MiCadena2;10)
MiCadena2:="Flores"
MiCadena1:=MiCadena2
```

La referencia del carácter no es válida.

```
i:=-30
MiCadena[[i]]:=MiCadena2 o MiCadena[[i]]:=MiCadena2
```

El parámetro es una cadena vacía.

```
MiCadena[[1]]:=""
MiCadena[[1]]:=""
```

Módulo por cero.

```
Var1:=0
Var2:=2
Var3:=Var2% Var1
```

Parámetros incorrectos en un comando Execute Formula.

```
EXECUTE FORMULA("MiMetodo(MiAlfa)")
```

si MiMetodo espera un parámetro diferente de un alfanumérico.

Puntero a una variable desconocida.

```
MiPuntero:=Get pointer("Variable")
MiPuntero:="MiCadena"
```

si Variable no aparece explícitamente en la base.
Intento de redireccionar utilizando un puntero.

```
Booleano:=Puntero->
```

si Puntero apunta a un campo de tipo Entero.
Mal uso de un puntero o puntero a una variable desconocida.

```
Caracter:=CadenaVar[[Puntero->]]
Caracter:=CadenaVar[[Puntero]]
```

si Puntero no apunta a un número.

C_BLOB

C_BLOB ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Descripción

C_BLOB asigna el tipo BLOB a cada variable especificada.

La primera sintaxis del comando, en el cual el parámetro opcional *método* NO se pasa, se utiliza para declarar una variable de proceso, interproceso o local.

Nota: esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando, en la cual el parámetro opcional *método* SÍ se pasa, se utiliza para declarar ante el compilador el resultado y/o los parámetros ($0, $1, $2 etc.) de un método. Utilice esta sintaxis del comando para evitar declarar las variables durante la compilación de una base, ahorrando tiempo de compilación.

Advertencia: la segunda sintaxis no se puede ejecutar en modo interpretado. Por esta razón, si utiliza esta sintaxis, mantengala en un método que no se ejecute en modo interpretado. El nombre de este método debe comenzar por "COMPILER."

Consejo avanzado: la sintaxis **C_BLOB(${...})** le permite declarar un número variable de parámetros del mismo tipo, bajo la condición de que estos sean los últimos parámetros del método. Por ejemplo, la declaración **C_BLOB(${5})** le indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de ese tipo. Para mayor información, consulte el comando **Count parameters.**

Ejemplos

Ver ejemplos en la sección .

C_BOOLEAN

C_BOOLEAN ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Descripción

El comando **C_BOOLEAN** asigna el tipo Booleano a cada variable especificada.

La primera sintaxis del comando, en la cual el parámetro opcional *método* NO se pasa, se utiliza para declarar una variable proceso, interproceso, o local.

Nota: esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando, en la cual el parámetro opcional *método* SI se pasa, se utiliza para predeclarar ante el compilador el resultado y/o los parámetros ($0, $1, $2 etc.) de un método. Utilice esta sintaxis para evitar la fase de declaración de variables durante la compilación de la base, ahorrando tiempo de compilación.

Advertencia: la segunda sintaxis no se puede ejecutar en modo interpretado. Por esta razón, si utiliza esta sintaxis, mantengala en un método que no se ejecute en modo interpretado. El nombre de este método debe comenzar por "COMPILER."

Consejo avanzado: la sintaxis **C_BOOLEAN(${...})** le permite declarar un número variable de parámetros del mismo tipo, bajo la condición de que estos sean los últimos parámetros del método. Por ejemplo, la declaración **C_BOOLEAN(${5})** le indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de ese tipo. Para mayor información, consulte el comando **Count parameters.**

Ejemplos

Ver ejemplos en la sección .

C_COLLECTION

C_COLLECTION ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método
variable	Variable	➡	Nombre(s) de la variable(s) o parámetro(s) ${...} a declarar

Descripción

El comando **C_COLLECTION** asigna el tipo *Colección* a todas las variables que se especifican.

El tipo *Colección* es soportado por el lenguaje 4D a partir de la v16 R4. Las variables de este tipo contienen una lista ordenada de valores de atributo objeto de todo tipo, almacenados como un array JSON. Las colecciones se gestionan utilizando los comandos del tema **Objetos (Lenguaje)**.

Se utiliza la primera sintaxis del comando (cuando no se pasa el parámetro método) para declarar y digitar una variable de tipo local, proceso o interproceso. Esta sintaxis se puede utilizar en bases de datos interpretadas.

Se utiliza la segunda sintaxis del comando (cuando se pasa el parámetro *método*) para declarar de antemano el resultado del método y/o los parámetros ($0, $1, $2, etc.) para el compilador. Debe utilizar esta sintaxis si desea omitir la fase de escritura de variables para ahorrar tiempo cuando se compila la base de datos.

ATENCIÓN: no puede ejecutar la segunda sintaxis en modo interpretado. Por esta razón, cuando se utiliza esta sintaxis, tiene que almacenar en un método (cuyo nombre debe comenzar por "COMPILER") que no se ejecuta en modo interpretado.

Uso avanzado: puede utilizar la sintaxis **C_COLLECTION**(${...}) para declarar un número variable de parámetros del mismo tipo para un método siempre y cuando sean los últimos parámetros de este método. Por ejemplo, la declaración **C_COLLECTION**(${5}) indica al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de este tipo.

Ejemplo

Usted desea declarar una variable proceso colección y llenarla con una nueva colección:

```
C_COLLECTION(myCol)
 //aquí el valor de myCol es nulo
myCol:=New collection("Green";100;"Orange";200;"Red";300)
 //myCol= ["Green",100,"Orange",200,"Red",300]
```

C_DATE

C_DATE ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Descripción

El comando **C_DATE** asigna el tipo Fecha a cada variable especificada.

La primera sintaxis del comando, en la cual el parámetro opcional *método* NO se pasa, se utiliza para declarar una variable proceso, interproceso, o local.

Nota: esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando, en la cual el parámetro opcional *método* SI se pasa, se utiliza para predeclarar ante el compilador el resultado y/o los parámetros ($0, $1, $2 etc.) de un método. Utilice esta sintaxis para evitar la fase de declaración de variables durante la compilación de la base, ahorrando tiempo de compilación.

Advertencia: la segunda sintaxis no se puede ejecutar en modo interpretado. Por esta razón, si utiliza esta sintaxis, mantengala en un método que no se ejecute en modo interpretado. El nombre de este método debe comenzar por "COMPILER."

Consejo avanzado: la sintaxis **C_DATE(${...})** le permite declarar un número variable de parámetros del mismo tipo, con la condición de que sean los últimos parámetros del método. Por ejemplo, la declaración **C_DATE(${5})** le indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de ese tipo. Para mayor información, consulte el comando **Count parameters**.

Ejemplos

Ver ejemplos en la sección .

C_LONGINT

C_LONGINT ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Descripción

El comando **C_LONGINT** asigna el tipo Entero largo a cada variable especificada.

La primera sintaxis del comando, en la cual el parámetro opcional *método* NO se pasa, se utiliza para declarar una variable proceso, interproceso, o local.

Nota: esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando, en la cual el parámetro opcional *método* SI se pasa, se utiliza para predeclarar ante el compilador el resultado y/o los parámetros ($0, $1, $2 etc.) de un método. Utilice esta sintaxis para evitar la fase de declaración de variables durante la compilación de la base, ahorrando tiempo de compilación.

Advertencia: la segunda sintaxis no se puede ejecutar en modo interpretado. Por esta razón, si utiliza esta sintaxis, mantengala en un método que no se ejecute en modo interpretado. El nombre de este método debe comenzar por "COMPILER."

Consejo avanzado: la sintaxis **C_LONGINT(${...})** le permite declarar un número variable de parámetros del mismo tipo, con la condición de que sean los últimos parámetros del método. Por ejemplo, la declaración **C_LONGINT(${5})** le indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de ese tipo. Para mayor información, consulte el comando **Count parameters.**

Ejemplos

Ver ejemplos en la sección .

C_OBJECT

C_OBJECT ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método
variable	Variable	➡	Nombre(s) de la(s) variable(s) o parámetro(s) ${...} a declarar

Descripción

El comando **C_OBJECT** asigna el tipo *Objeto* a todas las variables especificadas.

El tipo *Objeto* es soportado por el lenguaje 4D a partir de la v14. Los objetos de este tipo son administrados por los comandos del tema **Objetos (Lenguaje)** o por medio de la notación de objetos (ver **Uso de la notación objeto**).

La primera sintaxis del comando (si el parámetro *metodo* no se pasa) se utiliza para declarar y digitar una variable proceso, interproceso o local. Esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando (si el parámetro *metodo* se pasa) se utiliza para declarar de antemano el resultado y/o los parámetros ($0, $1, $2, etc.) de un método para el compilador. Debe utilizar esta sintaxis si desea evitar la fase de escribir variables con el fin de ahorrar tiempo al compilar la base de datos.

ADVERTENCIA: no puede ejecutar la segunda sintaxis en modo interpretado. Por esta razón, cuando se utiliza esta sintaxis, debe guardarla en un método (cuyo nombre debe comenzar por "COMPILER") que no se ejecute en modo interpretado .

Uso avanzado: la sintaxis **C_OBJECT**(${...}) permite declarar para un método un número variable de parámetros del mismo tipo, siempre y cuando sean los últimos parámetros del método. Por ejemplo, la declaración **C_OBJECT**(${5}) indica al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de este tipo.

Importante: el comandoC_OBJECT no crea un objeto llamado *variable*. Si desea acceder a las propiedades del objeto usando la notación de objetos, primero debe inicializarlo usando el comando **New object**, de lo contrario se devuelve un error de sintaxis.

Ejemplo

```
C_OBJET($obj)  //La variable $obj se declara pero el objeto $obj no existe
$obj:=New object  //El objeto $obj se inicializa
$obj.prop:=42  //...y sus propiedades pueden ser accedidas
```

C_PICTURE

C_PICTURE ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Descripción

El comando **C_PICTURE** asigna el tipo Imagen a cada variable especificada.

La primera sintaxis del comando, en la cual el parámetro opcional *método* NO se pasa, se utiliza para declarar una variable proceso, interproceso, o local.

Nota: esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando, en la cual el parámetro opcional *método* SI se pasa, se utiliza para predeclarar ante el compilador el resultado y/o los parámetros ($0, $1, $2 etc.) de un método. Utilice esta sintaxis para evitar la fase de declaración de variables durante la compilación de la base, ahorrando tiempo de compilación.

Advertencia: la segunda sintaxis no se puede ejecutar en modo interpretado. Por esta razón, si utiliza esta sintaxis, mantengala en un método que no se ejecute en modo interpretado. El nombre de este método debe comenzar por "COMPILER."

Consejo avanzado: la sintaxis **C_PICTURE(${...})** le permite declarar un número variable de parámetros del mismo tipo, con la condición de que sean los últimos parámetros del método. Por ejemplo, la declaración **C_PICTURE(${5})** le indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de ese tipo. Para mayor información, consulte el comando **Count parameters**.

Ejemplos

Ver ejemplos en la sección .

C_POINTER

C_POINTER ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Descripción

El comando **C_POINTER** asigna el tipo Puntero a cada variable especificada.

La primera sintaxis del comando, en la cual el parámetro opcional *método* NO se pasa, se utiliza para declarar una variable proceso, interproceso o local.

Nota: esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando, en la cual el parámetro opcional *método* SI se pasa, se utiliza para predeclarar ante el compilador el resultado y/o los parámetros ($0, $1, $2 etc.) de un método. Utilice esta sintaxis para evitar la fase de declaración de variables durante la compilación de la base, ahorrando tiempo de compilación.

Advertencia: la segunda sintaxis no se puede ejecutar en modo interpretado. Por esta razón, si utiliza esta sintaxis, manténgala en un método que no se ejecute en modo interpretado. El nombre de este método debe comenzar por "COMPILER."

Consejo avanzado: la sintaxis **C_POINTER(${...})** le permite declarar un número variable de parámetros del mismo tipo, con la condición de que sean los últimos parámetros del método. Por ejemplo, la declaración **C_POINTER(${5})** le indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de ese tipo. Para mayor información, consulte el comando **Count parameters**.

Ejemplos

Ver ejemplos en la sección .

C_REAL

C_REAL ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Descripción

El comando **C_REAL** asigna el tipo Real a cada variable especificada.

La primera sintaxis del comando, en la cual el parámetro opcional *método* NO se pasa, se utiliza para declarar una variable proceso, interproceso o local.

Nota: esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando, en la cual el parámetro opcional *método* SI se pasa, se utiliza para predeclarar ante el compilador el resultado y/o los parámetros ($0, $1, $2 etc.) de un método. Utilice esta sintaxis para evitar la fase de declaración de variables durante la compilación de la base, ahorrando tiempo de compilación.

Advertencia: la segunda sintaxis no se puede ejecutar en modo interpretado. Por esta razón, si utiliza esta sintaxis, manténgala en un método que no se ejecute en modo interpretado. El nombre de este método debe comenzar por "COMPILER."

Consejo avanzado: la sintaxis **C_REAL(${...})** le permite declarar un número variable de parámetros del mismo tipo, con la condición de que sean los últimos parámetros del método. Por ejemplo, la declaración **C_REAL(${5})** le indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de ese tipo. Para mayor información, consulte el comando **Count parameters.**

Ejemplos

Ver ejemplos en la sección .

C_TEXT

C_TEXT ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Descripción

El comando **C_TEXT** asigna el tipo Texto a cada variable especificada.

La primera sintaxis del comando, en la cual el parámetro opcional *método* NO se pasa, se utiliza para declarar una variable proceso, interproceso, o local.

Nota: esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando, en la cual el parámetro opcional *método* SI se pasa, se utiliza para predeclarar ante el compilador el resultado y/o los parámetros ($0, $1, $2 etc.) de un método. Utilice esta sintaxis para evitar la fase de declaración de variables durante la compilación de la base, ahorrando tiempo de compilación.

Advertencia: la segunda sintaxis no se puede ejecutar en modo interpretado. Por esta razón, si utiliza esta sintaxis, manténgala en un método que no se ejecute en modo interpretado. El nombre de este método debe comenzar por "COMPILER."

Consejo avanzado: la sintaxis **C_TEXT(${...})** le permite declarar un número variable de parámetros del mismo tipo, con la condición de que sean los últimos parámetros del método. Por ejemplo, la declaración **C_TEXT(${5})** le indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de ese tipo. Para mayor información, consulte el comando **Count parameters**.

Ejemplos

Ver ejemplos en la sección .

C_TIME

C_TIME ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Descripción

El comando **C_TIME** asigna el tipo Hora a cada variable especificada.

La primera sintaxis del comando, en la cual el parámetro opcional *método* NO se pasa, se utiliza para declarar una variable proceso, interproceso, o local.

Nota: esta sintaxis puede utilizarse en bases interpretadas.

La segunda sintaxis del comando, en la cual el parámetro opcional *método* SI se pasa, se utiliza para predeclarar ante el compilador el resultado y/o los parámetros ($0, $1, $2 etc.) de un método. Utilice esta sintaxis para evitar la fase de declaración de variables durante la compilación de la base, ahorrando tiempo de compilación.

Advertencia: la segunda sintaxis no se puede ejecutar en modo interpretado. Por esta razón, si utiliza esta sintaxis, manténgala en un método que no se ejecute en modo interpretado. El nombre de este método debe comenzar por "COMPILER."

Consejo avanzado: la sintaxis **C_TIME(${...})** le permite declarar un número variable de parámetros del mismo tipo, con la condición de que sean los últimos parámetros del método. Por ejemplo, la declaración **C_TIME(${5})** le indica a 4D y al compilador que a partir del quinto parámetro, el método puede recibir un número variable de parámetros de ese tipo. Para mayor información, consulte el comando **Count parameters**.

Ejemplos

Ver ejemplos en la sección .

IDLE

IDLE
Este comando no requiere parámetros

Descripción

El comando **IDLE** está diseñado para utilizarse únicamente con el compilador. Este comando se utiliza sólo en bases compiladas en las cuales los métodos definidos por el usuario se escriben de manera que no se hacen llamados al motor de 4D. Por ejemplo, si un método tiene un bucle **For** en el cual no se ejecutan comandos 4D, el bucle no puede ser interrumpido por un proceso instalado por **ON EVENT CALL**, y el usuario tampoco puede cambiar a otra aplicación. En este caso, debe insertar **IDLE** para permitir que 4D intercepte los eventos. Si no quiere interrupciones, omita **IDLE**.

Ejemplo

En el siguiente Ejemplo, el bucle no terminaría nunca en una base compilada sin llamar a **IDLE**:

```
  ` Método de proyecto Hacer algo
ON EVENT CALL("METODO EVENTO")
◇vbParar:=False
MESSAGE("Procesando..."+Char(13)+"Presione cualquier tecla para interrumpir...")
Repeat
  ` Hacer algún proceso que no involucre un comando 4D
   IDLE
Until(◇vbParar)
ON EVENT CALL("")
```

con:

```
  ` Método de proyecto METODO EVENTO
If(Undefined(KeyCode))
   KeyCode:=0
End if
If(KeyCode#0)
   CONFIRM("¿Está seguro de querer detener esta operación?")
   If(OK=1)
      ◇vbParar:=True
   End if
End if
```

_o_C_GRAPH

_o_C_GRAPH ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Cadena	➡	Nombre del método
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Nota de compatibilidad

Las variables del tipo área de gráfico son obsoletas y no se soportan desde 4D v14. Es necesario utilizar variables imagen (ver **GRAPH**).

_o_C_INTEGER

_o_C_INTEGER ({metodo ;} variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método opcional
variable		➡	Nombre de la(s) variable(s) a declarar

Nota preliminar

El comando **_o_C_INTEGER** se conserva en 4D por razones de compatibilidad con bases de datos antiguas. De hecho, 4D y el compilador declaran internamente a los Enteros como Enteros largos. Por ejemplo:

```
_o_C_INTEGER($MyVar)
$TheType:=Type($MyVar)  //$TheType = 9 (Is longint)
```

_o_C_STRING

_o_C_STRING ({metodo ;} tamaño ; variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
metodo	Método	➡	Nombre del método
tamaño	Entero largo	➡	Tamaño de la cadena
variable	Variable	➡	Nombre de la(s) variable(s) a declarar

Nota de compatibilidad

El funcionamiento del comando **[#current_title]** es estrictamente idéntico al del comando **C_TEXT** (el parámetro *tamaño* se ignora). Ahora se recomienda utilizar sólo **C_TEXT** en sus desarrollos 4D.

Comunicaciones

- GET SERIAL PORT MAPPING
- RECEIVE BUFFER
- RECEIVE PACKET
- RECEIVE RECORD
- RECEIVE VARIABLE
- SEND PACKET
- SEND RECORD
- SEND VARIABLE
- SET CHANNEL
- SET TIMEOUT
- USE CHARACTER SET

GET SERIAL PORT MAPPING

GET SERIAL PORT MAPPING (arrNumeros ; arrNombres)

Parámetro	Tipo		Descripción
arrNumeros	Array entero largo	⇐	Array de números de puertos
arrNombres	Array cadena	⇐	Array de nombres de puertos

Descripción

El comando **GET SERIAL PORT MAPPING** devuelve dos arrays, *arrNumeros* y *arrNombres*, que contienen los números y nombres de puertos seriales del equipo actual.

Este comando es útil en Mac OS X, donde el sistema operativo asigna dinámicamente el número de puerto cuando utiliza un adaptador serial USB. Puede direccionar cualquier puerto serial extendido utilizando su nombre (estático), sin importar su número actual.

Nota: este comando no devuelve valores significativos con puertos estándar. Si quiere direccionar un puerto estándar, debe pasar su valor (0 o 1) directamente utilizando el comando **SET CHANNEL** (modo antiguo de funcionamiento de 4D).

Ejemplo

Este método de proyecto puede utilizarse para direccionar el mismo puerto serial (sin protocolo), sin importar el número que se le haya asignado:

```
ARRAY TEXT($arrNombrePuertos;0)
ARRAY LONGINT($arrNumsPuertos;0)
C_LONGINT($vNumPuerto;$vNumPuertoFinal)

  `Buscar los números actuales de los puertos seriales
GET SERIAL PORT MAPPING($arrNumsPuertos;$arrNombrePuertos)
$vPortNum:=Find in array($arrNombrePuertos;vNombrePuerto)
  ` vNombrePuerto contiene el nombre del puerto a utilizar; puede venir de una ventana,
  ` de un valor almacenado en un campo, etc.
If(arrNumsPuertos{$vNumPuerto}=0)
    vNumPuertoFinal:=0  `caso especial sobre Mac OS X
else
    vNumPuertoFinal:=arrNumsPuertos{$vNumPuerto}+100
End if
SET CHANNEL(vNumPuertoFinal;params)  `params contiene los parámetros de comunicación
...  `Efectuar las operaciones deseadas
SET CHANNEL(11)  `Cierre del puerto
```

RECEIVE BUFFER

RECEIVE BUFFER (varRecep)

Parámetro	Tipo		Descripción
varRecep	Variable texto	➡	Variable para recibir datos

Descripción

RECEIVE BUFFER lee los datos del puerto serial abierto previamente por el comando **SET CHANNEL**. El puerto serial tiene un buffer que se llena con caracteres hasta que un comando los lee. **RECEIVE BUFFER** obtiene los caracteres del buffer serial, los coloca en la variable *varRecep* y luego limpia el buffer. Si no hay caracteres en el buffer, la variable *varRecep* estará vacía.

En Windows
El buffer de puerto serial en Windows está limitado en tamaño a 10 Kbytes. Esto significa que el buffer puede saturarse. Cuando está lleno y se reciben nuevos caracteres, los nuevos caracteres se reemplazan los antiguos caracteres. Los antiguos caracteres se pierden; por lo tanto, es esencial que el buffer se lea rápidamente cuando se reciben nuevos caracteres.

En Mac OS
El buffer del puerto serial en Mac OS X tiene una capacidad en principio ilimitada (depende de la memoria disponible). Si el buffer está lleno y se reciben nuevos caracteres, los nuevos caracteres reemplazan los antiguos caracteres. Los antiguos caracteres se pierden; por lo tanto, es esencial que el buffer se lea rápidamente cuando se reciben nuevos caracteres.

El comando **RECEIVE BUFFER** es diferente de **RECEIVE PACKET** en la medida en que recupera todo lo que encuentra en el buffer y lo devuelve inmediatamente. **RECEIVE PACKET** espera hasta encontrar un carácter específico o un cierto número de caracteres en el buffer.

Durante la ejecución de **RECEIVE BUFFER**, el usuario puede interrumpir la recepción presionando **Ctrl-Alt-Mayús** (Windows) o **Comando-Opción-Mayús** (Macintosh). Esta interrupción genera un error -9994 que puede interceptar con la ayuda de un método instalado por el comando **ON ERR CALL**.

Ejemplo

El método de proyecto **ESCUCHAR PUERTO SERIAL** utiliza **RECEIVE BUFFER** para obtener texto del puerto serial y acumularlo en una variable interproceso:

```
  ` ESCUCHAR PUERTO SERIAL
  ` Abrir el puerto serial
SET CHANNEL(201;Speed 9600+Data bits 8+Stop bits one+Parity none)
<>IP_Escuchar_Puerto_Serial:=True
While(<>IP_Escuchar_Puerto_Serial)
   RECEIVE BUFFER($vtBuffer)
   If((Length($vtBuffer)+Length(<>vtBuffer))>MAXTEXTLEN)
      <>vtBuffer:=""
   End if
   <>vtBuffer:=<>vtBuffer+$Buffer
End while
```

En este punto, cualquier otro proceso puede leer la variable interproceso *vtBuffer* para trabajar con los datos que vienen del puerto serial.

Para dejar de escuchar al puerto serial, ejecute:

```
  ` Dejar de escuchar al puerto serial
◇IP_Escuchar_Puerto_Serial:=False
```

Note que el acceso a la variable interproceso *vtBuffer* debe estar protegido por un semáforo, de manera que los procesos no entren en conflicto. Para mayor información ver el comando **Semaphore**.

RECEIVE PACKET

RECEIVE PACKET ({docRef ;} varRecep ; stopCar | numBytes)

Parámetro	Tipo		Descripción
docRef	DocRef	➡	Número de referencia del documento o canal actual (puerto serial o documento)
varRecep	Variable texto, BLOB variable	⬅	Variable para recibir datos
stopCar | numBytes	Cadena, Entero largo	➡	Carácter(es) en el(los) cual(es) detener la recepción de datos o número de bytes a recibir

Descripción

RECEIVE PACKET lee los caracteres desde un puerto serial o desde un documento.

Si se especifica *docRef*, el comando recupera los caracteres desde un documento abierto por la función **Open document**, **Create document** o **Append document**. Si se omite *docRef*, este comando recupera los caracteres de un puerto serial o de un documento abierto utilizando **SET CHANNEL**.

Sin importar la fuente, los caracteres leídos se devuelven en la variable *varRecep*, la cual debe ser una variable de tipo Texto, Alfa o BLOB. Si los caracteres han sido enviados por el comando **SEND PACKET**, el tipo debe corresponder al del paquete enviado.

Notas:

- Cuando el paquete recibido es de tipo BLOB, el comando no tiene en cuenta ningún conjunto de caracteres definido por el comando **USE CHARACTER SET.** Se devuelve el BLOB sin ninguna modificación.
- Cuando el paquete recibido es de tipo texto, el comando **RECEIVE PACKET** soporta Byte Order Marks (BOMs). En este caso, si el conjunto de caracteres actual es de tipo Unicode (UTF-8, UTF-16 o UTF-32), 4D intenta identificar un BOM entre los primeros bytes recibidos. Si detecta uno, lo filtra de la variable *recepVar* y 4D utiliza el conjunto de caracteres definido en lugar del conjunto de caracteres actual.

Para leer un número particular de caracteres, pase este número en el parámetro *numBytes.* Si la variable *recepVar* es de tipo Texto, en una sola llamada puede leer hasta 2 GB de texto (valor teórico).

Para recibir datos hasta una cadena particular (compuesta por uno o más caracteres) pase esta cadena en *stopCar* (la cadena no se devuelve en *varRecep*).

En este caso, si no se encuentra el carácter cadena especificado por *stopCar*:

- Cuando **RECEIVE PACKET** lee datos en un documento, se detendrá la lectura al final del documento.
- Cuando **RECEIVE PACKET** lee datos de un puerto serial, el comando se ejecutará indefinidamente hasta que el timeout (si hay) haya pasado (ver **SET TIMEOUT**) o hasta que el usuario interrumpa la recepción (ver a continuación).

Durante la ejecución de **RECEIVE PACKET**, el usuario puede interrumpir la recepción presionando Ctrl-Alt-Mayús (Windows) o Comando-Opción-Mayús (Macintosh). Esta interrupción genera un error -9994 que puede interceptar con un método instalado utilizando **ON ERR CALL**. Generalmente, sólo tendrá que administrar la interrupción de una recepción en el momento de una comunicación sobre un puerto serial.

Durante la lectura de un documento, el primer **RECEIVE PACKET** comienza por leer el principio del documento. La lectura de cada paquete subsiguiente comienza con el carácter después del último carácter leído.

Nota: este comando es útil con un documento abierto con **SET CHANNEL**. Por el contrario, para un documento abierto con **Open document**, **Create document** o **Append document** puede utilizar los comandos **Get document position** y **SET DOCUMENT POSITION** para obtener y cambiar la ubicación en el documento donde ocurrirá la próxima escritura (**SEND PACKET**) o lectura (**RECEIVE PACKET**).

Cuando intente leer después del final de un archivo **RECEIVE PACKET** devuelve los datos leídos hasta ese punto y la variable OK tomará el valor 1. Luego, el siguiente **RECEIVE PACKET** devolverá una cadena vacía y la variable OK tomará el valor cero.

Nota: en modo no Unicode (compatibilidad) cuando utiliza el comando **RECEIVE PACKET** para leer caracteres de un documento Windows y no quiere utilizar mapas ASCII para convertir los caracteres Windows en caracteres Mac OS, puede utilizar la función **Win to Mac**.

Ejemplo 1

El siguiente ejemplo lee 20 caracteres de un puerto serial en la variable *recupVeinte*:

```
RECEIVE PACKET(recupVeinte;20)
```

Ejemplo 2

El siguiente ejemplo lee datos del documento referenciado por la variable *miDoc* en la variable *vData*. El comando lee hasta que encuentra un retorno de carro:

```
RECEIVE PACKET(miDoc;vData;Char(Carriage return))
```

Ejemplo 3

El siguiente ejemplo lee datos desde el documento referenciado por la variable *miDoc* en la variable *vData*. El comando lee hasta que encuentra una etiqueta HTML de fin de tabla *</TD>*:

```
RECEIVE PACKET(miDoc;vData;"</TD>")
```

Ejemplo 4

El siguiente ejemplo lee datos de un documento y los pone en campos. Los datos se almacenan como campos de longitud fija. El método llama a una subrutina para eliminar espacios (espacios al final de la cadena). La subrutina sigue el método:

```
$vhDocRef :=Open document("";"TEXT")  ` Apertura de un documento de tipo TEXTO
If(OK=1)  ` Si el documento está abierto
   Repeat  ` Bucle hasta que no haya más datos
      RECEIVE PACKET($vhDocRef;$Var1;15)  ` Lectura de 15 caracteres
      RECEIVE PACKET($vhDocRef;$Var2;15)  ` Hace lo mismo para el segundo campo
      If(($Var1#"")|($Var2#""))  ` Si por lo menos uno de los campos no está vacío
         CREATE RECORD([Personas])  ` Crear un nuevo registro
         [Personas]Nombre:=Strip($Var1)  ` Guardar el nombre
         [Personas]Apellido:=Strip($Var2)  ` Guardar el apellido
         SAVE RECORD([Personas])  ` Guardar el registro
      End if
   Until(OK=0)
   CLOSE DOCUMENT($vhDocRef)  ` Cierre del documento
End if
```

Los espacios al final de los datos son eliminados por el siguiente método, llamado **Elimina**:

```
For($i;Length($1);1;-1)  ` Bucle desde el final de la cadena al inicio
   If($1[[$i]]#" ")  ` Si no es un espacio···
      $i :=-$i  ` Forzar el bucle a detenerse
   End if
End for
$0:=Delete string($1;-$i;Length($1))  ` Borrar los espacios
```

Variables y conjuntos del sistema

Después de un llamado a **RECEIVE PACKET**, la variable sistema OK toma el valor 1 si el paquete se recibe sin errores. De lo contrario, la variable sistema OK toma el valor 0.

RECEIVE RECORD

RECEIVE RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla en la cual recibir el registro o Tabla por defecto, si se omite

Descripción

RECEIVE RECORD recibe un registro en *tabla* del puerto serial o de un documento abierto por el comando **SET CHANNEL**. El registro debe haber sido envíado con **SEND RECORD**. Cuando ejecuta **RECEIVE RECORD**, se crea automáticamente un nuevo registro en *tabla*. Si el registro se recibe correctamente, entonces debe utilizar **SAVE RECORD** para guardar el nuevo registro.

Se recibe el registro completo. Esto significa que también se reciben todos los subregistros, imágenes y BLOBs almacenados en el registro.

Importante: cuando los registros se envía y reciben utilizando **SEND RECORD** y **RECEIVE RECORD**, la estructura de la tabla fuente y la estructura de la tabla de destino deben ser compatibles. Si no lo son, 4D convertirá los valores de acuerdo a las definiciones de las tablas cuando se ejecute **RECEIVE RECORD**.

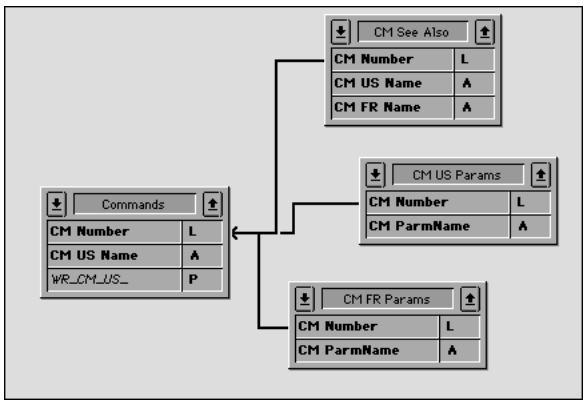
Notas:

1. Si recibe un registro de un documento utilizando este comando, el documento debe haber sido abierto utilizando el comando **SET CHANNEL**. No puede utilizar **RECEIVE RECORD** con un documento abierto con **Open document**, **Append document** o **Create document**.
2. Durante la ejecución de **RECEIVE RECORD**, el usuario puede interrumpir la recepción presionando Ctrl-Alt-Mayús (Windows) o Comando-Opción-Mayús (Macintosh). Esta interrupción genera un error -9994 que puede interceptar con el método instalado por el comando **ON ERR CALL**. Generalmente, sólo debe manejar la interrupción de una recepción durante una comunicación sobre puerto serial.

Ejemplo

El uso combinado de **SEND VARIABLE**, **SEND RECORD**, **RECEIVE VARIABLE** y **RECEIVE RECORD** es ideal para archivar datos o intercambiar datos entre bases monopuesto idénticas utilizada en diferentes lugares. Puede intercambiar datos entre bases 4D utilizando los comandos de importación/exportación como **EXPORT TEXT** y **IMPORT TEXT**. Sin embargo, si sus datos contienen imágenes o tablas relacionadas, utilizar **SEND RECORD** y **RECEIVE RECORD** es mucho más conveniente.

Por ejemplo, la documentación que está leyendo ha sido creada utilizando 4D y 4D Write. Como varios escritores en diferentes lugares del mundo estaban trabajando en este proyecto, necesitábamos una forma simple de intercambiar datos entre las diferentes bases de datos. Esta es una vista simplificada de la estructura de la base:



La tabla *[Commands]* contiene la descripción de cada comando o sección. Las tablas *[CM US Params]* y *[CM FR Params]* contienen respectivamente los parámetros de cada comandos en Inglés y Francés. La tabla *[CM See also]* contiene los comandos indicados como referencias para cada comando o sección. El intercambio de la documentación entre las bases consiste en enviar los registros de *[Commands]* así como sus registros relacionados. Para hacerlo, utilizamos **SEND RECORD** y **RECEIVE RECORD**. Adicionalmente, utilizamos **SEND VARIABLE** y **RECEIVE VARIABLE** para marcar los documentos de importación/exportación con etiquetas.

Este es el método de proyecto (simplificado) para exportar la documentación:

```
  ` Método de proyecto CM_EXPORT_SEL
  ` Este método funciona con la selección actual de la tabla [Commands]

SET CHANNEL(12;"")  ` Permitimos al usuario crear y abrir un documento channel
If(OK=1)
  ` Marcamos el documento con un variable que indique su contenido
  ` Nota: la variable de proceso BUILD_LANG indica si los datos US (Inglés) o FR (Francés) son enviados
   $vsTag:="4DV6COMMAND"+BUILD_LANG
   SEND VARIABLE($vsTag)
  ` Enviar una variable indicando cuántos [Commands] son enviados
   $vlNbCmd:=Records in selection([Commands])
   SEND VARIABLE($vlNbCmd)
   FIRST RECORD([Commands])
```

```
    ` Para cada comando
    For($vlCmd;1;$vlNbCmd)
  ` Enviar el registro [Commands]
      SEND RECORD([Comands])
  ` Selección de todos los registros seleccionados
      RELATE MANY([Commands])
  ` Dependiendo del lenguaje, enviar una variable indicando
  ` el número de parámetros que sigue
      Case of
        :(BUILD_LANG="US")
          $vlNbParm:=Records in selection([CM US Params])
        :(BUILD_LANG="FR")
          $vlNbParm:=Records in selection([CM FR Params])
      End case
      SEND VARIABLE($vlNbParm)
  ` Enviar los registros de los parámetros (si hay)
      For($vlParm;1;$vlNbParm)
        Case of
          :(BUILD_LANG="US")
            SEND RECORD([CM US Params])
            NEXT RECORD([CM US Params])
          :(BUILD_LANG="FR")
            SEND RECORD([CM FR Params])
            NEXT RECORD([CM FR Params])
        End case
      End for
  ` Enviar una variable indicando cuántos “See Also” siguen
      $vlNbSee:=Records in selection([CM See Also])
      SEND VARIABLE($vlNbSee)
  ` Enviar los registros [See Also] (si hay)
      For($vlSee;1;$vlNbSee)
        SEND RECORD([CM See Also])
        NEXT RECORD([CM See Also])
      End for
  ` Ir al siguiente registro [Commands] y continuar la exportación
      NEXT RECORD([Commands])
    End for
    SET CHANNEL(11) ` Cerrar el documento
End if
```

Este es el método de proyecto (simplificado) de importación de la documentación:

```
  ` Método de proyecto CM_IMPORT_SEL

SET CHANNEL(10;"") ` Permitimos al usuario abrir un documento existente
If(OK=1) ` Si un documento estaba abierto
  RECEIVE VARIABLE($vsTag) ` Tratamos de recibir la variable marcada esperada
  If($vsTag="4DV6COMMAND@") ` ¿Recibimos la etiqueta correcta?
    $CurLang:=Substring($vsTag;Length($vsTag)-1) ` Extraemos el lenguaje de la etiqueta
    If(($CurLang="US") | ($CurLang="FR")) ` ¿Recibimos un lenguaje válido
      RECEIVE VARIABLE($vlNbCmd) ` ¿Cuántos comandos hay en este documento?
      If($vlNbCmd>0) ` Si hay al menos uno
        For($vlCmd;1;$vlNbCmd) ` Para cada registro [Commands] archivado
  ` Recepción del registro
          RECEIVE RECORD([Commands])
  ` Llamar una subrutina que guarde el nuevo registro o copie sus valores
  ` en un registro existente
          CM_IMP_CMD($CurLang)
  ` Recibir el número de parámetros (si hay)
          RECEIVE VARIABLE($vlNbParm)
          If($vlNbParm>=0)
  ` Llamar una subrutina que llama RECEIVE RECORD luego guarda los nuevos registros
  ` o los copia en registros existentes
            CM_IMP_PARM($vlNbParm;$CurLang)
          End if
  ` Recepción dle número de “See Also” (si hay)
          RECEIVE VARIABLE($vlNbSee)
          If($vlNbSee>0)
  ` Llamar una subrutina que llame RECEIVE RECORD luego guarde los nuevos registros
  ` o los copies en registros existentes
            CM_IMP_SEEA($vlNbSee;$CurLang)
```

```
                End if
            End for
        Else
            ALERT("El número de comandos en este documento de exportación es inválido.")
        End if
    Else
        ALERT("El lenguaje de este documento de exportación es desconocido.")
    End if
Else
    ALERT("Este documento NO es un documento de exportación de comandos.")
End if
SET CHANNEL(11) ` Cerrar documento
End if
```

Note que no probamos la variable OK mientras recibimos los datos ni tratamos de interceptar los eventuales errores. Sin embargo, como almacenamos variables en el documento que describe el documento en sí mismo, si estas variables, una vez recibidas, tienen sentido, la probabilidad de error es muy baja. Si por ejemplo un usuario abre mal un documento, la primera prueba detiene la operación de inmediato.

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si se recibe el registro. De lo contrario, toma el valor 0.

RECEIVE VARIABLE

RECEIVE VARIABLE (variable)

Parámetro	Tipo		Descripción
variable	Variable	⇐	Variable en la cual recibir

Descripción

RECEIVE VARIABLE recibe a *variable*, la cual fue enviada previamente por **SEND VARIABLE** desde el documento o puerto serial previamente abierto por **SET CHANNEL**.

En modo interpretado, si la variable no existe antes del llamado de **RECEIVE VARIABLE**, la variable se crea, digita y asigna de acuerdo a lo que se ha recibido. En modo compilado, la variable debe ser del mismo tipo de la que se recibe. En ambos casos, el contenido de la variable se reemplaza con el de la variable recibida.

Notas:

1. Si recibe una variable desde un documento utilizando este comando, el documento debe haber sido abierto utilizando el comando **SET CHANNEL**. No puede utilizar **RECEIVE VARIABLE** con un documento abierto con **Open document**, **Append document** o **Create document**.

2. Este comando no soporta variables de tipo array. Si quiere enviar y recibir arrays desde un documento o un puerto serial, utilice los **Comandos BLOB**.

3. Durante la ejecución de **RECEIVE VARIABLE**, el usuario puede interrumpir la recepción presionando Ctrl-Alt-Mayús (Windows) o Comando-Opción-Mayús (Macintosh). Esta interrupción genera un error -9994 que puede interceptar con un método instalado por el comando **ON ERR CALL**. Generalmente, sólo necesita administrar la interrupción de una recepción mientras la comunicación sobre un puerto serial.

Ejemplo

Ver el ejemplo del comando **RECEIVE RECORD**.

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si se recibe la variable. De lo contrario, la variable sistema OK toma el valor 0.

SEND PACKET

SEND PACKET ({docRef ;} paquete)

Parámetro	Tipo		Descripción
docRef	DocRef	⇒	Número de referencia del documento o canal actual (puerto serial o documento)
paquete	Cadena, BLOB	⇒	Cadena o BLOB a enviar

Descripción

El comando **SEND PACKET** envía un paquete a un puerto serial o a un documento. Si *docRef* se especifica, el paquete se escribe en el documento referenciado por *docRef*. Si *docRef* no se especifica, el paquete se escribe para el puerto serial o documento previamente abierto por el comando **SET CHANNEL**.

Un *paquete* es sólo una serie simple de datos, generalmente una cadena de caracteres.

También puede pasar un BLOB en *paquete*. Esto le permite evitar las restricciones relacionadas con la codificación de los caracteres enviados en modo texto (ver ejemplo 2).

Nota: cuando pasa un BLOB en *paquete*, el comando no tiene en cuenta ningún conjunto de caracteres definido por el comando **USE CHARACTER SET**. El BLOB se envía sin ninguna modificación.

Antes de utilizar **SEND PACKET**, debe abrir un puerto serial o un documento con **SET CHANNEL**, o abrir un documento con uno de los comandos de gestión de documentos.

Cuando escribe un documento, el primer **SEND PACKET** comienza a escribir al comienzo del documento a menos que el documento hubiera sido abierto con **USE CHARACTER SET**. Hasta el momento en que el documento se cierra, cada paquete subsiguiente se añade a los paquetes enviados anteriormente.

Nota: este comando es útil para un documento abierto con **SET CHANNEL**. Por otra parte, para un documento abierto con **Open document**, **Create document** o **Append document**, usted puede utilizar los comandos **Get document position** y **SET DOCUMENT POSITION** para obtener y cambiar la ubicación en el documento donde ocurrirá la próxima escritura (**SEND PACKET**) o lectura (**RECEIVE PACKET**).

Ejemplo 1

El siguiente ejemplo escribe datos de campos en un documento. Escribe los campos como campos de longitud fija. Si un campo es de longitud menor a la especificada, el campo se llena con espacios. (es decir, se añaden espacios para alcanzar la longitud especificada.) Aunque el uso de campos de longitud fija es un método ineficiente de almacenamiento de datos, algunos sistemas informáticos y ciertas aplicaciones todavía lo utilizan:

```
$vhDocRef :=Create document("")  ` Creación de un documento
If(OK=1)  ` ¿Se creó el documento?
   For($vlRegistro;1;Records in selection([Personas]))  ` Bucle para cada registro
  ` Envío de un paquete creado a partir de una cadena de 15 espacios que contiene el campo Nombre
      SEND PACKET($vhDocRef;Change string(15*Char(SPACE);[Personas]Nombre;1))
  ` Envío de un segundo paquete creado a partir de una cadena de 15 espacios que contiene el campo Apellido
  ` Este podría estar en el primer SEND PACKET, pero se separa por claridad
      SEND PACKET($vhDocRef;Change string(15*Char(SPACE);[Personas]Apellido;1))
      NEXT RECORD([Personas])
   End for
  ` Envío de Char(26), utilizado como marcador de fin de registro por algunos ordenadores
   SEND PACKET($vhDocRef;Char(SUB ASCII code))
   CLOSE DOCUMENT($vhDocRef)  ` Cierre del documento
End if
```

Ejemplo 2

Este ejemplo ilustra el envío y recuperación de caracteres extendidos vía un BLOB en un documento:

```
C_BLOB($enviar_blob)
C_BLOB($recibir_blob)
TEXT TO BLOB("âzértÿ";$enviar_blob;UTF8 text without length)
SET BLOB SIZE($enviar_blob;16;255)
$enviar_blob{6}:=0
$enviar_blob{7}:=1
$enviar_blob{8}:=2
$enviar_blob{9}:=3
$enviar_blob{10}:=0
$vlDocRef:=Create document("blob.test")
If(OK=1)
   SEND PACKET($vlDocRef;$enviar_blob)
   CLOSE DOCUMENT($vlDocRef)
End if
```

```
$vlDocRef</Gen9>=<Gen9>Open document(document)
If(OK=1)
    RECEIVE PACKET($vlDocRef;$recibir_blob;65536)
    CLOSE DOCUMENT($vlDocRef)
End if
```

SEND RECORD

SEND RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual enviar el registro actual o tabla por defecto, si se omite

Descripción

SEND RECORD envía el registro actual de *tabla* al puerto serial o a un documento abierto por el comando **SET CHANNEL**. El registro se envía con un formato interno especial que sólo puede ser leído por **RECEIVE RECORD**. Si no existe un registro actual, **SEND RECORD** no tiene efecto.

Se envía el registro completo. Esto significa que todos los subregistros, imágenes y BLOBs almacenados en el registro también son enviados.

Importante: cuando los registros se envían y recuperan utilizando **SEND RECORD** y **RECEIVE RECORD**, la estructura de la tabla fuente y la estructura de la tabla de destino deben ser compatibles. Si no lo son, 4D convertirá los valores de acuerdo a las definiciones de la tabla cuando se ejecute **RECEIVE RECORD**.

Nota: si envía un registro a un documento utilizando este comando, el documento debe haber sido abierto utilizando el comando **SET CHANNEL**. No es posible utilizar **SEND RECORD** con un documento abierto con **Open document**, **Append document** o **Create document**.

Nota de compatibilidad: a partir de la versión 11 de 4D este comando no soporta subtablas.

Ejemplo

Ver el ejemplo del comando **RECEIVE RECORD**.

SEND VARIABLE

SEND VARIABLE (variable)

Parámetro	Tipo		Descripción
variable	Variable	➡	Variable a enviar

Descripción

SEND VARIABLE envía *variable* al documento o puerto serial previamente abierto por **SET CHANNEL**. La variable se envía con un formato interno especial que puede ser leído sólo por **RECEIVE VARIABLE**. **SEND VARIABLE** envía la variable completa (incluyendo su tipo y valor).

Notas:

1. Si envía una variable a un documento utilizando este comando, el documento debe haber sido abierto por el comando **SET CHANNEL**. No puede utilizar **SEND VARIABLE** con un documento abierto con **Open document**, **Append document** o **Create document**.
2. Este comando no soporta variables de tipo Array. Si quiere enviar y recibir arrays de un documento o un puerto serial, utilice los nuevos *comandos BLOB* introducidos en la versión 6.

Ejemplo

Ver el ejemplo del comando **RECEIVE RECORD**.

SET CHANNEL

SET CHANNEL (puerto ; param)

Parámetro	Tipo		Descripción
puerto	Entero largo	➡	Número de puerto serial
param	Entero largo	➡	Parámetros de puerto serial

SET CHANNEL (operacion ; doc)

Parámetro	Tipo		Descripción
operacion	Entero largo	➡	Operación a efectuar en el documento
doc	Cadena	➡	Nombre del documento

Descripción

El comando **SET CHANNEL** abre un puerto serial o un documento. Sólo puede abrir un puerto serial o un documento al tiempo con este comando. Para cerrar un puerto serial abierto, pase SET CHANNEL (11).

Nota histórica: este comando fue originalmente el primer comando 4D utilizado para trabajar con puertos seriales y documentos en discos. Desde entonces, se han añadido nuevos comandos. Hoy en día, se trabaja generalmente con documentos en disco utilizando los comandos **Open document**, **Create document** y **Append document**. Con estos comandos, puede leer y escribir caracteres en los documentos utilizando **SEND PACKET** o **RECEIVE PACKET** (estos comandos trabajan también con **SET CHANNEL**). Sin embargo, si quiere utilizar los comandos **SEND VARIABLE**, **RECEIVE VARIABLE**, **SEND RECORD** y **RECEIVE RECORD**, debe utilizar **SET CHANNEL** para acceder a los documentos en el disco.

La descripción de **SET CHANNEL** está compuesta de dos secciones:

- Trabajar con los puertos seriales
- Trabajar con documentos

Trabajar con los puertos seriales - SET CHANNEL (puerto;parametros)

La primera forma del comando **SET CHANNEL** abre un puerto serial, define el protocolo de comunicación así como otra información del puerto. Los datos pueden ser enviados por los comandos **SEND PACKET**, **SEND RECORD** o **SEND VARIABLE**, y recibidos con **RECEIVE BUFFER**, **RECEIVE PACKET**, **RECEIVE RECORD** o **RECEIVE VARIABLE**.

- El primer parámetro, *puerto*, selecciona el puerto y el protocolo. Puede direccionar hasta 99 puertos seriales (uno a la vez). La siguiente tabla lista los valores para *puerto*:

Valores puerto	Descripción
0	Puerto impresora (Macintosh) o COM2 (PC) sin protocolo
1	Puerto modem (Macintosh) o COM1 (PC) sin protocolo
20	Puerto impresora (Macintosh) o COM2 (PC) con protocolo de sofware tal como XON/XOFF
21	Puerto modem (Macintosh) o COM1 (PC) con protocolo de sofware tal como XON/XOFF
30	Puerto impresora (Macintosh) o COM2 (PC) con protocolo de hardware tal como RTS/CTS
31	Puerto modem (Macintosh) o COM1 (PC) con protocolo de hardware tal como RTS/CTS
101 a 199	Comunicación serial sin protocolo
201 a 299	Comunicación serial con protocolo de software tal como XON/XOFF
301 a 399	Comunicación serial con protocolo de hardware tal como RTS/CTS

Importante: el valor que pasa en *puerto* se refiere a un puerto serial COM existente reconocido por el sistema operativo. Por ejemplo, para que pueda utilizar los valores 101, 103 y 125, los puertos seriales COM1, COM3 y COM25 deben haber sido configurados correctamente.

Nota sobre los puertos seriales

En una configuración estándar, Mac OS y Windows reconocen dos puertos seriales: en Mac OS, el puerto modem y el puerto impresora; en Windows, los puertos COM1 y COM2. Sin embargo, se pueden añadir puertos seriales adicionales por medio de tableros de extensión. Inicialmente, 4D sólo direccionaba dos puertos seriales estándar y más adelante se implementó el soporte de puertos adicionales. Por razones de compatibilidad, se conservan ambos sistemas de direccionamiento.
- Si quiere direccionar un puerto serial estándar (impresora/COM2 o modem/COM1), puede pasar en el parámetro *puerto* uno de los siguientes valores 0, 1, 20, 21, 30 y 31 (que corresponde al método de direccionamiento antiguo), o un valor mayor a 100 (por favor vea la siguiente explicación).
- Si quiere direccionar puertos seriales adicionales, debe pasar el valor N+100 (donde N es el valor del puerto a direccionar). También puede considerar añadir 100 o 200 al valor menciondo anteriormente (N+100), si quiere seleccionar respectivamente un protocolo de software o de hardware.

Ejemplo 1

Si quiere utilizar el puerto impresora/COM2 sin protocolo, puede utilizar una de las siguientes sintaxis:

```
SET CHANNEL(0;param)
```

o

```
SET CHANNEL(102;param)
```

Ejemplo 2

Si quiere utilizar el puerto modem/COM1 con el protocolo XON/XOFF, puede utilizar una de las siguientes sintaxis:

```
SET CHANNEL(21;param)
```

o

```
SET CHANNEL(201;param)
```

Ejemplo 3

Si quiere utilizar el puerto COM 25 con el protocolo RTS/CTS, debe utilizar las siguientes sintaxis:

```
SET CHANNEL(325;param)
```

• El parámetro *param* fija la velocidad, el número de bits de datos, el número de bits de stop y la paridad. Puede determinar el valor para los *param* añadiendo los valores de velocidad, bits de datos, bits de stop y paridad, como se listan en la siguiente tabla. Por ejemplo, para definir 1200 baud, 8 bits de datos, 1 bit de stop y ninguno de paridad, usted sumaría 94 + 3072 + 16384 + 0 = 19550. Entonces pasaría 19550 como el valor del parámetro *param.*

	Valor a acumular en param	**Descripción**
Velocidad	380	300
(en baud)	189	600
	94	1200
	62	1800
	46	2400
	30	3600
	22	4800
	14	7200
	10	9600
	4	19200
	2	28800
	1	38400
	0	57600
	1022	115200
	1021	230400
Bits de datos	0	5
	2048	6
	1024	7
	3072	8
Bits de stop	16384	1
	–32768	1.5
	–16384	2
Paridad	0	Ninguno
	4096	Impar
	12288	Par

Truco: los diferentes valores numéricos a acumular y pasar en *puerto* y *param* (a excepción de los valores de COM1...COM99) están disponibles como constantes predefinidas en el tema **Comunicaciones** del explorador en el entorno Diseño. Para los valores COM1...COM99, utilice los valores numéricos literales.

Trabajar con documentos en disco - SET CHANNEL(operacion;documento)

La segunda forma del comando **SET CHANNEL** le permite crear, abrir, y cerrar un documento. A diferencia de los comandos del tema Documentos del sistema, **SET CHANNEL** sólo puede abrir un documento a la vez. El documento puede leerse o escribirse. Consulte la sección para mayor información al respecto.

El parámetro *operacion* especifica la operación a realizar en el documento especificado por *documento*. La siguientes tabla lista los valores de *operacion* y el resultado obtenido, en función del valor de *documento*. La primera columna lista los valores posibles de *operacion*. La segunda columna lista los valores posibles de *documento*. La tercera columna lista la operación resultante.

Por ejemplo, para visualizar una caja de diálogo de abrir un archivo, puede utilizar la siguiente línea:

```
SET CHANNEL(13;"")
```

Operación	Documento	Resultado
10	Cadena	Abre el documento especificado por Cadena. Si el documento no existe, se crea y abre el documento.
10	"" (cadena vacía)	Muestra la ventana de abrir archivo. Todos los tipos de archivos se presentan.
11	ninguno	Cierra un archivo abierto.
12	"" (cadena vacía)	Muestra la ventana de guardar archivo para crear un nuevo archivo.
13	"" (cadena vacía)	Muestra la ventana de abrir archivo. Sólo se presentan los archivos de tipo texto.

Todas las operaciones en esta tabla modifican la variable sistema **Document** si es necesario. Igualmente la variable sistema OK toma el valor 1 si la operación fue exitosa. De lo contrario, la variable sistema OK toma el valor 0.

Ejemplo 4

Ver los ejemplos de los comandos **RECEIVE BUFFER**, **SET TIMEOUT** y **RECEIVE RECORD**.

SET TIMEOUT

SET TIMEOUT (segundos)

Parámetro	Tipo		Descripción
segundos	Entero largo	➡	Número de segundos hasta el timeout

Descripción

El comando **SET TIMEOUT** especifica cuánto tiempo tiene para la ejecución un comando de puerto serial. Si el comando no termina dentro del tiempo especificado, *segundos*, el comando del puerto serial se cancela, se genera un error -9990, y la variable sistema OK toma el valor 0. Puede interceptar este error con la ayuda de un método instalado por el comando **ON ERR CALL**.

Note que el tiempo es el tiempo total permitido para que el comando se ejecute, no el tiempo entre los caracteres recibidos. Para cancelar un parámetro anterior y detener el monitoreo de la comunicación de puerto serial, utilice un parámetro de 0 para *segundos*.

Los comandos que se afectan por el parámetro timeout son:

- **RECEIVE PACKET**
- **RECEIVE RECORD**
- **RECEIVE VARIABLE**

Ejemplo

El siguiente ejemplo fija el puerto serial para recibir datos y el timeout. Los datos se leen con **RECEIVE PACKET**. Si los datos no se reciben en el tiempo definido, ocurre un error:

```
SET CHANNEL(MacOS serial port;Speed 9600+Data bits 8+Stop bits one+Parity none) ` Apertura del puerto serial
SET TIMEOUT(10) ` Fijar el timeout en 10 segundos
ON ERR CALL("INTERCEPTAR ERRORES COM") ` Tratar las interrupciones eventuales
RECEIVE PACKET(vtBuffer;Char(13)) ` Leer hasta encontrar un retorno de carro
If(OK=0)
   ALERT("Error durante la recepción de datos.")
Else
   [Personas]Nombre:=vtBuffer ` Guardar los datos recibidos en un campo
End if
ON ERR CALL("")
```

USE CHARACTER SET

USE CHARACTER SET (mapa {; mapaImpExp})

Parámetro	Tipo		Descripción
mapa	Cadena, Operador	➡	Nombre del conjunto de caracteres a a utilizar (Modo Unicode) o nombre del documento del mapa ASCII a utilizar (Modo ASCII) o * para restaurar el mapa ASCII/conjunto de caracteres por defecto
mapaImpExp	Entero largo	➡	0 = Mapa de exportación 1 = Mapa de importación Si se omite, mapa de exportación

Descripción

USE CHARACTER SET modifica el conjunto de caracteres utilizado por 4D para todas las operaciones de transferencia de datos entre la base y un documento o puerto serial para el proceso actual. Las operaciones de transferencia incluyen la importación y exportación de texto, DIF y SYLK. Un mapa de caracteres también funciona con los datos enviados por los comandos **SEND PACKET**, **RECEIVE PACKET** (para paquetes de tipo texto), y **RECEIVE BUFFER**. No tiene efecto en transferencias de datos realizadas con **SEND RECORD**, **SEND VARIABLE**, **RECEIVE RECORD**, **SEND PACKET**, **RECEIVE PACKET** (para paquetes tipo BLOB) y **RECEIVE VARIABLE**.
El parámetro *mapa* debe corresponder al nombre "IANA" del conjunto de caracteres a utilizar, o a uno de sus alias. Por ejemplo, los nombres "iso-8859-1" o "utf-8" son ambos nombres válidos, así como los alias "latin1" u "11". Para mayor información sobre estos nombres, por favor consulte la siguiente dirección: *http://www.iana.org/assignments/character-sets*. También se presentan ejemplos de nombres IANA en la descripción del comando **CONVERT FROM TEXT.**

Si *mapaImpExp* es 0, el mapa está definido para la exportación. Si *mapaImpExp* es 1, el mapa es para importación. Si no pasa el parámetro *mapaImpExp*, se utiliza el mapa de exportación por defecto.

Cuando se pasa el parámetro *, el conjunto de caracteres por defecto se restablece (mapa de importación o exportación dependiendo del valor de *mapaImpExp.*

En 4D, el conjunto de caracteres por defecto es UTF-8.

Ejemplo

El siguiente ejemplo (modo Unicode) utiliza el conjunto de caracteres UTF-16 para exportar un texto, luego restablece el conjunto de caracteres por defecto:

```
USE CHARACTER SET("UTF-16LE";0) ` Utilizar el conjunto de caracteres UTF-16 "Litttle Endian"
EXPORT TEXT([MiTabla];"MiTexto") ` Exportar los datos con el mapa
USE CHARACTER SET(*;0) ` Restablecer el conjunto de caracteres por defecto
```

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si el mapa se carga correctamente, de lo contrario toma el valor 0.

Conjuntos

- Conjuntos
- ADD TO SET
- CLEAR SET
- COPY SET
- CREATE EMPTY SET
- CREATE SET
- CREATE SET FROM ARRAY
- DIFFERENCE
- INTERSECTION
- Is in set
- LOAD SET
- Records in set
- REMOVE FROM SET
- SAVE SET
- UNION
- USE SET

Conjuntos

Los conjuntos son una forma poderosa y rápida de manipular selecciones de registros. Además de la posibilidad de crear conjuntos, asociarlos a la selección actual, guardarlos, cargarlos y borrarlos, 4D permite realizar tres operaciones estándar de conjuntos:

- Intersección
- Unión
- Diferencia.

Conjuntos y selección actual

Un conjunto es una representación de una selección de registros. La idea de conjuntos está íntimamente ligada a la idea de selección actual. Los conjuntos generalmente se utilizan por las siguientes razones:

- Para guardar y luego restaurar una selección cuando el orden no tiene importancia
- Para acceder a la selección que el usuario realiza en pantalla (el conjunto UserSet)
- Para realizar una operación lógica entre las selecciones.

La selección actual es una lista de referencias que apuntan a cada registro actualmente seleccionado. La lista existe en memoria. Sólo los registros seleccionados están en la lista. Una selección no contiene realmente los registros, sólo una lista de referencias a los registros. Cada referencia a un registro utiliza 4 bytes en memoria. Cuando usted trabaja sobre una tabla, siempre trabaja con los registros en la selección actual. Cuando una selección está ordenada, sólo la lista de referencias se reorganiza. Sólo hay una selección actual para cada tabla en un proceso.

Como una selección actual, un conjunto representa una selección de registros. Un conjunto utiliza una representación muy compacta para cada registro. Cada registro está representado por un bit (un octavo de byte). Las operaciones que utilizan conjuntos son muy rápidas, porque los computadores pueden realizar muy rápidamente operaciones sobre bits. Un conjunto contiene un bit por cada registro de la tabla, sin importar si el registro está incluido o no en el conjunto. De hecho, cada bit es igual a 1 o 0, dependiendo de si el registro está en el conjunto o no.

Los conjuntos son muy económicos en términos de memoria RAM. El tamaño de un conjunto, en bytes, siempre es igual al número total de registros en la tabla dividido por 8. Por ejemplo, si crea un conjunto para una tabla que contiene 10 000 registros, el conjunto toma 1 250 bytes, aproximadamente 1.2K en RAM.

Pueden existir varios conjuntos para cada tabla. De hecho, los conjuntos pueden guardarse en el disco independientemente de la base. Para cambiar un registro que pertenece a un conjunto, primero debe utilizar el conjunto como selección actual, luego modificar el registro.

Un conjunto nunca está ordenado, los registros simplemente son marcados como que pertenecen al conjunto o no. Por el contrario, una selección temporal está ordenada, pero requiere más memoria en la mayoría de los caso. Para mayor información sobre selecciones temporales, consulte la sección **Selecciones temporales**.

En el momento de su creación, un conjunto "recuerda" el registro actual de la selección. La siguiente tabla compara los conceptos de la selección actual y de los conjuntos:

Comparación	**Selección actual**	**Conjuntos**
Número por tabla	1	ilimitado
Ordenable	Sí	No
Puede guardarse en disco	No	Sí
RAM por registro (en bytes)	Número de registros selec.* 4	Número total de registros/8
Combinable	No	Sí
Contiene registro actual	Sí	Sí, como el del momento de la creación

El conjunto pertenece a la tabla en la cual se ha creado. Las operaciones sobre conjuntos sólo pueden realizarse entre los conjuntos que pertenecen a la misma tabla.

Los conjuntos son independientes de los datos, esto significa que después de realizar modificaciones a una tabla, un conjunto podría no ser exacto. Hay muchas operaciones que pueden hacer que un conjunto no sea exacto. Por ejemplo, si crea un conjunto de todos los habitantes de Madrid y cambia los datos de uno de los registros por Barcelona, el conjunto no cambia, porque el conjunto es simplemente la representación de una selección de registros. La eliminación y el remplazo de registros también pueden volver el conjunto obsoleto, al igual que la compactación de datos. La exactitud de los conjuntos sólo se puede garantizar si los datos de la selección original no han cambiado.

Conjuntos proceso e interproceso

Puede utilizar tres tipos de conjuntos:

- **Conjuntos proceso**: un conjunto proceso sólo es accesible por el proceso en el cual fue creado. **LockedSet** es un conjunto proceso. Los conjuntos proceso se borran tan pronto como el método de proceso termina. Los conjuntos proceso no necesitan un prefijo especial en el nombre.
- **Conjuntos interproceso**: un conjunto es interproceso si el nombre del conjunto está precedido por los símbolos (<>), un signo "menor que" seguido por un signo "mayor que". **Nota**: esta sintaxis puede utilizarse en Windows y Macintosh. Igualmente, en Macintosh, puede utilizar el diamante (Opción-Mayús-V).
  Un conjunto interproceso es "visible" para todos los procesos de la base.
  En modo cliente/servidor, un conjunto interproceso es "visible" para todos los procesos de la máquina donde fue creado (cliente o servidor).
  El nombre de un conjunto interproceso debe ser único en la base.

- **Conjuntos locales/cliente**: los conjuntos locales/cliente son principalmente para uso en modo cliente/servidor. El nombre de los conjuntos local/cliente está precedido por el signo dólar ($), excepto para el conjunto sistema **UserSet**. A diferencia de otros tipos de conjuntos, un conjunto local/cliente se almacena en el equipo cliente.

Notas:

- El tamaño máximo de un nombre de conjunto es de 255 caracteres (excluyendo <> y los símbolos $).
- Para mayor información sobre la utilización de conjuntos en modo cliente/servidor, consulte la sección **4D Server, conjuntos y selecciones temporales** del Manual 4D Server.

Visibilidad de Conjuntos

La siguiente tabla indica los principios de visibilidad de los conjuntos en función de su alcance y de dónde fueron creados:

	Procesos Cliente	Otros Procesos Cliente	Otros Clientes	Procesos Servidor	Otros Procesos Servidor
Creación de un proceso cliente					
$prueba	x				
prueba	x			x(Trigger)	
<>prueba	x	x			
Creación de un proceso servidor					
$prueba				x	
prueba				x	
<>prueba				x	x

Conjuntos y transacciones

Un conjunto puede ser creado al interior de una transacción. Es posible crear un conjunto de registro creados dentro de una transacción y un conjunto de registros creados o modificados fuera de la transacción. Cuando la transacción termina, debe borrarse el conjunto creado durante la transacción, porque podría no ser una representación exacta de los registros, especialmente si la transacción se canceló.

Ejemplo de conjuntos

El siguiente ejemplo borra los registros duplicados de una tabla que contiene información de personas. Un bucle **For...End for** pasa por todos los registros, comparando el registro actual con el registro anterior. Si el nombre, dirección, y código postal son iguales, el registro se añade a un conjunto. Al final del bucle, el conjunto se convierte en la selección actual y la selección actual anterior se borra:

```
CREATE EMPTY SET([Personas];"Duplicados")
  ` Crear un conjunto vacío para los registros duplicados
ALL RECORDS([Personas])
  ` Seleccione todos los registros
  ` Ordenar los registros por código postal, dirección y nombre
  ` de manera que los duplicados estén juntos
ORDER BY([Personas];[Personas]CodigoPostal;>;[Personas]Direccion;>;[Personas]Nombre;>)
  ` Inicializar las variables que conservan los campos del registro anterior
$Nombre:=[Personas]Nombre
$Direccion:=[Personas]Direccion
$CodigoPostal:=[Personas]CodigoPostal
  ` Ir al segundo registro para compararlo con el primero
NEXT RECORD([Personas])
For($i;2;Records in table([Personas]))
  ` Bucle por los registros a partir de 2
  ` Si el nombre, dirección y código postal son los mismos
  ` del registro anterior, es un registro duplicado.
   If(([Personas]Nombre=$Nombre) & ([Personas]Direccion=$Direccion) &
      ([Personas]CodigoPostal=$CodigoPostal))
  ` Añadir registro actual (el duplicado) al conjunto
      ADD TO SET([Personas];"Duplicados")
   Else
  ` Guardar el nombre, dirección y código postal de este registro para compararlo con el siguiente
      $Nombre:=[Personas]Nombre
      $Direccion:=[Personas]Direccion
      $CodigoPostal:=[Personas]CodigoPostal
   End if
  ` Pasar al siguiente registro
   NEXT RECORD([Personas])
End for
  ` Utilizar los registros duplicados encontrados
USE SET("Duplicados")
  ` Borra los registros duplicados
DELETE SELECTION([Personas])
```

```
` Eliminar el conjunto de la memoria
CLEAR SET("Duplicados")
```

En lugar de borrar inmediatamente los registros al final del método, puede mostrarlos en pantalla o imprimirlos, de manera que pueda realizar una comparación más detallada.

The UserSet System Set

4D administra un conjunto sistema llamado *UserSet*, el cual guarda automáticamente las selecciones de registros más recientes realizadas por el usuario en pantalla. De esta forma, puede mostrar un grupo de registros con **MODIFY SELECTION** o **DISPLAY SELECTION**, pedirle al usuario seleccionar algunos y devolver el resultado de esta selección manual en un conjunto.

4D Server: aunque su nombre no comienza con el carácter "$", el conjunto sistema *UserSet* es un conjunto cliente. De manera que cuando utilice **INTERSECTION**, **UNION** y **DIFFERENCE**, asegúrese de comparar *UserSet* únicamente con otros conjuntos clientes. Para mayor información, por favor consulte las descripciones de estos comandos como también la sección **4D Server, conjuntos y selecciones temporales** del manual 4D Server.

Sólo hay un *UserSet* por proceso. Cada tabla no tiene su propio *UserSet*. *UserSet* se asocia a una tabla sólo cuando se muestra una selección de registros para la tabla.

4D administra el conjunto *UserSet* para los formularios listados mostrados en modo Diseño o utilizando los comandos **MODIFY SELECTION** o **DISPLAY SELECTION**. Sin embargo, este mecanismo no está activo para subformularios.

El siguiente método ilustra cómo mostrar registros para permitir al usuario seleccionar algunos registros, y luego utiliza *UserSet* para mostrar los registros seleccionados:

```
` Mostrar todos los registros y permitir al usuario seleccionar algunos de ellos.
` Luego muestre esta selección utilizando UserSet para cambiar la selección actual.
FORM SET OUTPUT([Personas];"Mostrar") ` Definir el formulario de salida
ALL RECORDS([Personas]) ` Selección de todas las personas
ALERT("Presione Ctrl o Comando y haga clic para seleccionar los registros.")
DISPLAY SELECTION([Personas]) ` Mostrar las personas
USE SET("UserSet") ` Utiliza las personas seleccionadas
ALERT("Usted eligió las siguientes personas.")
DISPLAY SELECTION([Personas]) ` Mostrar las personas seleccionadas
```

Conjunto sistema LockedSet

Los comandos **APPLY TO SELECTION**, **DELETE SELECTION**, **ARRAY TO SELECTION** y **JSON TO SELECTION** crean un conjunto sistema llamado **LockedSet** cuando se utilizan en un entorno multiproceso.
Los comandos de búsqueda crean igualmente un conjunto sistema *LockedSet* cuando encuentran registros bloqueados en el contexto de "búsqueda y bloqueo" (ver el comando **SET QUERY AND LOCK**).
LockedSet indica cuáles registros fueron bloqueados durante la ejecución del comando.

ADD TO SET

ADD TO SET ({tabla ;} conjunto)

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla del registro actual o Tabla por defecto si se omite
conjunto	Cadena	⇒	Nombre del conjunto al cual añadir el registro actual

Descripción

ADD TO SET añade el registro actual de *tabla* a *conjunto*. El conjunto ya debe existir; si no, ocurre un error. Si no hay un registro actual para *tabla*, **ADD TO SET** no tiene efecto.

CLEAR SET

CLEAR SET (conjunto)

Parámetro	Tipo		Descripción
conjunto	Cadena	➡	Nombre del conjunto a borrar de la memoria

Descripción

CLEAR SET borra *conjunto* de la memoria y libera la memoria utilizada por *conjunto*. **CLEAR SET** no afecta las tablas, selecciones, o registros. Para guardar un conjunto antes de borrarlo, utilice el comando **SAVE SET**. Como los conjuntos utilizan memoria, es bueno borrarlos cuando ya no se necesitan.

Ejemplo

Ver el ejemplo del comando **USE SET**.

COPY SET

COPY SET (srcCon ; dstCon)

Parámetro	Tipo		Descripción
srcCon	Cadena	➡	Nombre del conjunto fuente
dstCon	Cadena	➡	Nombre del conjunto de destino

Descripción

El comando **COPY SET** copia el contenido del conjunto *srcCon* en el conjunto *dstCon*.

Cada uno de estos conjuntos pueden ser procesos, interprocesos o de tipo local/cliente. Los dos conjuntos no tienen que ser del mismo tipo (como se muestra en los ejemplos a continuación), siempre y cuando ambos sean visibles en la máquina. Para más información acerca de este punto, consulte "**Visibilidad de Conjuntos**".

Ejemplo 1

El siguiente ejemplo, copia el conjunto "**ConjuntoA",** en el conjunto *"ConjuntoB"*:

```
COPY SET("ConjuntoA";"ConjuntoB")
```

Ejemplo 2

El siguiente ejemplo en Cliente/Servidor, copia el conjunto proceso "**ConjuntoA",** conservado en el equipo servidor, en el conjunto local *"$ConjuntoB",* conservado en el equipo cliente:

```
COPY SET("ConjuntoA";"$ConjuntoB")
```

CREATE EMPTY SET

CREATE EMPTY SET ({tabla ;} conjunto)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual crear un conjunto vacío o Tabla por defecto si se omite
conjunto	Cadena	➡	Nombre del nuevo conjunto vacío

Descripción

CREATE EMPTY SET crea un conjunto vacío, *conjunto*, para *tabla*. Puede añadir registros a este conjunto con el comando **ADD TO SET**. Si ya existe un conjunto con el mismo nombre, el conjunto existente se borra y reemplaza con el nuevo conjunto.

Nota: no tiene que utilizar **CREATE EMPTY SET** antes de utilizar **CREATE SET**.

Ejemplo

Consulte el ejemplo de la sección **Conjuntos**.

CREATE SET

CREATE SET ({tabla ;} conjunto)

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla para la cual crear un conjunto a partir de la selección actual o Tabla por defecto si se omite
conjunto	Cadena	⇒	Nombre del nuevo conjunto

Descripción

CREATE SET crea un nuevo conjunto, *conjunto*, para *tabla*, y coloca la selección actual en *conjunto*. El puntero del registro actual de la tabla se guarda con *conjunto*. Si *conjunto* se utiliza con **USE SET**, la selección actual y el registro actual se restauran. Como para todo conjunto, no hay criterio de ordenación; cuando *conjunto* se utiliza, se utiliza el orden por defecto. Si ya existe un conjunto con el mismo nombre, el conjunto existente se borra y reemplaza por el nuevo conjunto.

Ejemplo

El siguiente ejemplo crea un conjunto después de efectuar una búsqueda, de manera que el conjunto pueda guardarse en el disco:

```
QUERY([Personas]) ` El usuario efectúa una búsqueda
CREATE SET([Personas];"ConjuntoBusqueda") ` Creación de un nuevo conjunto
SAVE SET("ConjuntoBusqueda";"MiBusqueda") ` El conjunto se guarda en el disco
```

CREATE SET FROM ARRAY

CREATE SET FROM ARRAY (tabla ; arrayReg {; nomConjunto})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del conjunto
arrayReg	Entero largo, Array booleano	➡	Array de número de registros o Array de booleanos (True = el registro está en el conjunto, False = el registro no está en el conjunto)
nomConjunto	Cadena	➡	Nombre del conjunto a crear o Aplicar el comando a Userset si se omite

Descripción

El comando **CREATE SET FROM ARRAY** crea *nomCon* a partir de:

- Un array de número de registros absolutos *arrayReg* de la tabla *tabla*,
- o un array de booleanos *arrayReg*. En este caso, los valores del array indican si cada registro en la tabla pertenece (**True**) o no (**False**) a *nomCon*.

Cuando utilice este comando y pasa un array entero largo en *arrayReg*, todos los números en el array representan la lista de números de registros que está en *nomCon*. Si un número es inválido (por ejemplo, si un registro no ha sido creado), se genera el error -10503.

Cuando utilice este comando y pase un array booleano en *arrayReg*, el elemento N del array indica si el registro "N" está (**True**) o no (**False**) en *nomCon*. En principio, el número de elementos del array debe ser igual al número de registros en la tabla. Si el array es más pequeño que el número de registros, sólo los registros definidos por el array estarán en el conjunto.

Nota: con un array de booleanos, el comando utiliza los elementos 0 a N-1.

Si no pasa el parámetro *nomCon* o si pasa una cadena vacía, el comando se aplica al conjunto sistema **Userset**.

Gestión de errores

En un array de enteros largos, si un número de registro no es válido (registro no creado), se genera el error -10503.

DIFFERENCE

DIFFERENCE (conjunto1 ; conjunto2 ; resultado)

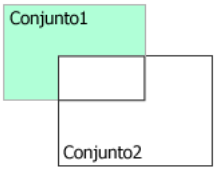
Parámetro	Tipo		Descripción
conjunto1	Cadena	➡	Conjunto inicial
conjunto2	Cadena	➡	Conjunto a restar
resultado	Cadena	➡	Conjunto resultante

Descripción

DIFFERENCE fusiona *conjunto1* y *conjunto2* y excluye del conjunto *resultado* todos los registros de *conjunto2*. En otras palabras, un registro se incluye en el *resultado* sólo si está en *conjunto1*, pero no en *conjunto2*. La siguiente tabla muestra todos los resultados posibles de una operación de diferencia de conjuntos.

Conjunto1	Conjunto2	Conjunto resultante
Sí	No	Sí
Sí	Sí	No
No	Yes	No
No	No	No

El esquema a continuación representa gráficamente una operación de diferencia entre dos conjuntos. El área de color es el conjunto resultante.



El conjunto *resultado* se crea por **DIFFERENCE**. El conjunto *resultado* reemplaza todo conjunto que exista con el mismo nombre, incluyendo *conjunto1* y *conjunto2*. Los conjuntos *conjunto1* y *conjunto2* deben ser de la misma tabla. El conjunto *resultado* pertenece a la misma tabla que *conjunto1* y *conjunto2*.

4D Server: en modo cliente/Servidor, los conjuntos interprocesos y procesos se conservan en el equipo servidor, mientras que los conjuntos locales se mantiene en los equipos cliente. **DIFFERENCE** requiere que los tres conjuntos estén en el mismo equipo. Por lo tanto, todos los conjuntos deben ser locales o ninguno de ellos debe ser local. Para mayor información consulte *4D Server and Sets* en el manual de referencia de 4D Server.

Ejemplo

Este ejemplo excluye los registros seleccionados por el usuario. Los registros se muestran en la pantalla con la siguiente instrucción:

```
DISPLAY SELECTION([Clientes]) ` Visualización de los clientes en una lista
```

Al final de la lista de registros hay un botón con un método de objeto. El método de objeto excluye los registros seleccionados por el usuario (el conjunto sistema llamado "UserSet"), y muestra una selección reducida:

```
CREATE SET([Clientes];"$Actual") ` Creación de un conjunto a partir de la selección actual
DIFFERENCE("$Actual";"UserSet";"$Actual") ` Exclusión de los registros seleccionados
USE SET("$Actual") ` Utilización del nuevo conjunto
CLEAR SET("$Actual") ` Borrar el conjunto
```

INTERSECTION

INTERSECTION (conjunto1 ; conjunto2 ; resultado)

Parámetro	Tipo		Descripción
conjunto1	Cadena	➡	Primer conjunto
conjunto2	Cadena	➡	Segundo conjunto
resultado	Cadena	➡	Conjunto resultante

Descripción

INTERSECTION compara *conjunto1* y *conjunto2* y selecciona únicamente los registros que están en ambos conjuntos. La siguiente tabla lista todos los resultados posibles de una operación de intersección de conjuntos.

Conjunto1	Conjunto2	Conjunto resultante
Sí	No	No
Sí	Sí	Sí
No	Sí	No
No	No	No

El resultado gráfico de una operación de intersección se muestra a continuación. El área de color el el conjunto resultante.

El conjunto *resultado* se crea por **INTERSECTION**. El conjunto *resultado* reemplaza todo conjunto que exista con el mismo nombre, incluyendo *conjunto1* y *conjunto2*. Los conjuntos *conjunto1* y *conjunto2* deben ser de la misma tabla. El conjunto *resultado* pertenece a la misma tabla que *conjunto1* y *conjunto2*. Si el mismo registro actual se define en *conjunto1* y *conjunto2*, permanece memorizado en *resultado*. De lo contrario, *resultado* no tiene un registro actual.

4D Server: en modo cliente/Servidor, los conjuntos interprocesos y procesos se conservan en el equipo servidor, mientras que los conjuntos locales se mantiene en los equipos cliente. **INTERSECTION** requiere que los tres conjuntos estén en el mismo equipo. Por lo tanto, todos los conjuntos deben ser locales o ninguno de ellos debe ser local. Para mayor información consulte la sección **4D Server, conjuntos y selecciones temporales** en el manual de referencia de 4D Server.

Ejemplo

El siguiente ejemplo busca los clientes que son atendidos por dos representantes de ventas, Juan y Ariel. Cada representante de ventas tiene un conjunto con sus clientes. Los clientes que se encuentran en ambos conjuntos son los que están en contacto con Juan y Ariel:

```
INTERSECTION("Juan";"Ariel";"Ambos") ` Coloque los clientes de ambos conjuntos en Ambos
USE SET("Ambos") ` Utilización del conjunto
CLEAR SET("Ambos") ` Borrado de este conjunto pero se guardan los otros
DISPLAY SELECTION([Clientes]) ` Muestra los clientes en contacto con los dos representantes de ventas
```

Is in set

Is in set (conjunto) -> Resultado

Parámetro	Tipo		Descripción
conjunto	Cadena	➡	Nombre del conjunto a borrar
Resultado	Booleano	⮌	El registro actual está en el conjunto (True) o El registro actual no está en el conjunto (False)

Descripción

Is in set prueba si el registro actual de la tabla está en *conjunto*. La función **Is in set** devuelve TRUE si el registro actual de la tabla está en *conjunto*, y FALSE si el registro actual de la tabla no está en *conjunto*.

Ejemplo

El siguiente ejemplo es un método de objeto de un botón que prueba si el registro actual está en el conjunto de los mejores clientes:

```
If(Is in set("Mejores"))  ` Probar si es un buen cliente
    ALERT("Es uno de los mejores clientes.")
Else
    ALERT("No es uno de los mejores clientes.")
End if
```

LOAD SET

LOAD SET ({tabla ;} conjunto ; doc)

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla a la cual pertenece el conjunto o Tabla por defecto si se omite
conjunto	Cadena	⇒	Nombre del conjunto a crear en memoria
doc	Cadena	⇒	Documento que contiene el conjunto

Descripción

LOAD SET carga un conjunto desde el archivo *documento,* creado con el comando **SAVE SET**.

El conjunto guardado en *documento* debe aplicar a *tabla*. El conjunto creado en memoria se sobrescribe si ya existe.

El parámetro *documento* es el nombre del documento disco que contiene el conjunto. El documento no necesita tener el mismo nombre del conjunto. Si pasa una cadena vacía en *documento*, se muestra una caja de diálogo estándar de apertura de archivos, permitiendo al usuario elegir el conjunto a cargar.

Recuerde que un conjunto es una representación de una selección de registros en el momento en que el conjunto se crea. Si los registros representados por el conjunto cambian, el conjunto podría volverse inválido. Por lo tanto, un conjunto cargado desde un disco debe representar a un grupo de registros que no cambia con frecuencia. Múltiples eventos pueden volver un conjunto inválido: modificación o eliminación de un registro del conjunto, o modificación de los criterios que determinan la creación del conjunto.

Ejemplo

El siguiente ejemplo utiliza **LOAD SET** para cargar un conjunto de sedes de la empresa Acme en Nueva York:

```
LOAD SET([Empresas];"NY Acme";"NYAcmeSt") ` Cargar el conjunto en memoria
USE SET("NY Acme") ` Cambiar la selección actual a NY Acme
CLEAR SET("NY Acme") ` Borrar el conjunto de la memoria
```

Variables y conjuntos del sistema

Si el usuario hace clic en Cancelar en la caja de diálogo de abrir archivos, o si se produce un error durante la carga, la variable sistema OK toma el valor 0. De lo contrario, toma el valor 1.

Records in set

Records in set (conjunto) -> Resultado

Parámetro	Tipo		Descripción
conjunto	Cadena	→	Nombre del conjunto a probar
Resultado	Entero largo	←	Número de registros en prueba

Descripción

Records in set devuelve el número de registros en *conjunto*. Si *conjunto* no existe, o si no hay registros en *conjunto*, **Records in set** devuelve 0.

Ejemplo

El siguiente ejemplo muestra una alerta indicando el porcentaje de clientes que se consideran como los mejores:

```
  ` Calcular primero el porcentaje
$Porcentaje :=(Records in set("Mejores")/Records in table([Clientes]))*100
  ` Mostrar una alerta con el porcentaje
ALERT(String($Porcentaje ;"##0%")+" de nuestros clientes son los mejores.")
```

REMOVE FROM SET

REMOVE FROM SET ({tabla ;} conjunto)

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla del registro actual o Tabla por defecto si se omite
conjunto	Cadena	⇒	Nombre del conjunto del cual eliminar el registro actual

Descripción

REMOVE FROM SET elimina el registro actual de *tabla* de *conjunto*. El conjunto ya debe existir; si no existe, ocurre un error. Si no existe un registro actual para *Tabla*, **REMOVE FROM SET** no tiene efecto.

SAVE SET

SAVE SET (conjunto ; doc)

Parámetro	Tipo		Descripción
conjunto	Cadena	⇒	Nombre del conjunto a guardar
doc	Cadena	⇒	Nombre del archivo en el cual guardar el conjunto

Descripción

SAVE SET guarda *conjunto* en el archivo *documento*.

No es necesario que *documento* tenga el mismo nombre que el conjunto. Si pasa una cadena vacía en *documento*, aparece una caja de diálogo de guardar archivos de manera que el usuario pueda introducir el nombre del documento. Con el comando **LOAD SET** puede cargar un conjunto guardado.

Si el usuario hace clic en Cancelar en la caja de diálogo de guardar archivo, o si se presenta un error durante la operación de guardar, la variable sistema OK toma el valor 0. De lo contrario, toma el valor 1.

SAVE SET con frecuencia se utiliza para guardar en disco los resultados de una búsqueda larga.

Advertencia: recuerde que un conjunto es una representación de una selección de registros en el momento en que se crea el conjunto. Si los registros representados por el conjunto cambian, el conjunto podría volverse inválido. Por lo tanto, un conjunto guardado en disco debe representar a un grupo de registros que no cambia con frecuencia. Múltiples eventos pueden volver un conjunto inválido: modificación o eliminación de un registro del conjunto, o modificación de los criterios que determinan la creación del conjunto. Igualmente recuerde que los conjuntos no guardan valores de campos.

Ejemplo

El siguiente ejemplo muestra la caja de diálogo estándar de guardar archivos con el fin de permitir al usuario introducir el nombre del documento que contiene el conjunto:

```
SAVE SET("UnConjunto";"")
```

Variables y conjuntos del sistema

Si el usuario hace clic en el botón Cancelar en la caja de diálogo de guardar archivos, o si hay un error durante la operación de carga, la variable sistema OK toma el valor 0. De lo contrario, toma el valor 1.

UNION

UNION (conjunto1 ; conjunto2 ; resultado)

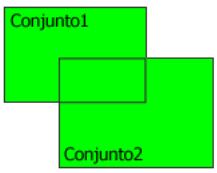
Parámetro	Tipo		Descripción
conjunto1	Cadena	➡	Primer conjunto
conjunto2	Cadena	➡	Segundo conjunto
resultado	Cadena	➡	Conjunto resultante

Descripción

UNION crea un nuevo conjunto que contiene todos los registros de *conjunto1* y *conjunto2*. La siguiente tabla muestra todos los posibles resultados de una operación de unión de conjuntos.

Conjunto1	Conjunto2	Conjunto resultante
Sí	No	Sí
Sí	Sí	Sí
No	Sí	Sí
No	No	No

El resultado de una operación de unión se muestra a continuación. La parte de color es el conjunto resultante.



El conjunto *resultado* se crea por **UNION**. El conjunto *resultado* reemplaza todo conjunto existente que tenga el mismo nombre, incluyendo *conjunto1* y *conjunto2*. Los conjuntos *conjunto1* y *conjunto2* deben ser de la misma tabla. El conjunto *resultado* pertenece a la misma tabla que *conjunto1* y *conjunto2*. El registro actual de *resultado* es el registro actual de *conjunto1*.

4D Server: en modo cliente/Servidor, los conjuntos interprocesos y procesos se conservan en el equipo servidor, mientras que los conjuntos locales se mantiene en los equipos cliente. **UNION** requiere que los tres conjuntos estén en el mismo equipo. Por lo tanto, todos los conjuntos deben ser locales o ninguno de ellos debe ser local. Para mayor información consulte *4D Server and Sets* en el manual de referencia de 4D Server.

Ejemplo

Este ejemplo añade registros al conjunto de mejores clientes. Los registros se muestran en la pantalla. Después de mostrar los registros en la pantalla, se carga un conjunto de los mejores clientes, y todos los registros seleccionados por el usuario (el conjunto sistema llamado "UserSet") se añaden al conjunto. Finalmente, el nuevo conjunto se guarda en el disco:

```
ALL RECORDS([Clientes])  ` Selección de todos los registros
DISPLAY SELECTION([Clientes])  ` Mostrar todos los clientes en una lista
LOAD SET("$Mejores";"$GuardarMejores")  ` Cargar el conjunto de los mejores clientes
UNION("$Mejores";"UserSet";"$Mejores")  ` Adición de la selección al conjunto
SAVE SET("$Mejores";"$GuardarMejores")  ` Guardar el conjunto de los mejores clientes
```

USE SET

USE SET (conjunto)

Parámetro	Tipo		Descripción
conjunto	Cadena	➡	Nombre del conjunto a utilizar

Descripción

USE SET crea, con los registros de *conjunto,* una nueva selección actual para la tabla a la cual pertenece *conjunto*.

Cuando crea un conjunto, la posición del registro actual se guarda. **USE SET** recupera esta información y hace del registro el nuevo registro actual. Si borra este registro antes de ejecutar **USE SET**, 4D selecciona como registro actual el primer registro del conjunto. Los comandos de Conjuntos **INTERSECTION**, **UNION**, **DIFFERENCE**, y **ADD TO SET** reinicializan el registro actual. Igualmente, si usted crea un conjunto que no contiene la posición del registro actual, **USE SET** selecciona el primer registro en el conjunto como registro actual.

Advertencia: recuerde que un conjunto es la representación de una selección de registros en un momento dado, el momento de la creación del conjunto. Si los registros que el conjunto representa se modifican, el conjunto podría no ser válido. Por lo tanto, un conjunto guardado en el disco debe representar a un grupo de registros que no cambia frecuentemente. Muchos eventos pueden invalidar la validez de un conjunto, como por ejemplo la supresión o modificación de un registro del conjunto, o la modificación de los criterios de creación del conjunto.

Ejemplo

El siguiente ejemplo utiliza **LOAD SET** para cargar un conjunto de sitios de la empresa Acme ubicada en Nueva York. Luego utiliza **USE SET** para hacer del conjunto la selección actual:

```
LOAD SET([Empresas];"NY Acme";"NYAcmeSt") ` Cargar el conjunto en memoria
USE SET("NY Acme") ` Cambiar la selección actual a NY Acme
CLEAR SET("NY Acme") ` Borrar el conjunto de la memoria
```

Control de entrada

- EDIT ITEM
- FILTER KEYSTROKE
- Get edited text
- GET HIGHLIGHT
- GOTO OBJECT
- HIGHLIGHT TEXT
- Keystroke

EDIT ITEM

EDIT ITEM ({* ;} objeto {; elemento})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una tabla o variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Tabla o variable (si se omite *)
elemento	Entero largo	➡	Número de elemento

Descripción

El comando **EDIT ITEM** le permite editar el elemento actual o el elemento de número *elemento* del Array o la lista designada por el parámetro *objeto*.

Esto significa que el elemento seleccionado puede modificarse; la entrada de un carácter reemplazará totalmente el contenido del elemento.

Si pasa el parámetro opcional *, indica que el parámetro *objeto* es un nombre de objeto (en este caso, pase una cadena en *objeto*). Si no pasa el parámetro, indica que el parámetro *objeto* es una tabla o una variable. En este caso, no pasa una cadena sino una referencia de una tabla o variable.

Este comando aplica a los siguientes objetos editables:

- Listas jerárquicas
- List boxes
- Subformularios (en este caso, sólo un nombre de objeto, el subformulario, puede pasarse en *objeto*),
- Formularios listados mostrados utilizando los comandos **MODIFY SELECTION** o **DISPLAY SELECTION**.

Si el comando se utiliza con un objeto editable que no está en la lista, actúa de la misma forma que el comando **GOTO OBJECT**. El comando no hace nada si la lista o el array están vacíos o son invisibles. Igualmente, si la lista o el array no son editables, el comando sólo selecciona el elemento especificado sin cambiar a modo edición. En el caso de las list boxes, si la columna no permite la entrada de texto (entrada por casillas de selección o por listas deplegables únicamente), el elemento especificado toma el foco.

El parámetro opcional *elemento* le permite designar la posición del elemento (lista jerárquica) o el número de línea (list box, formularios listados y subformulario en modo "selección múltiple") para cambiar a modo de edición. Si no pasa este parámetro, el comando se aplica al elemento actual de *objeto*. Si no hay elemento actual, el primer elemento de *objeto* cambia a modo edición.

Notas:

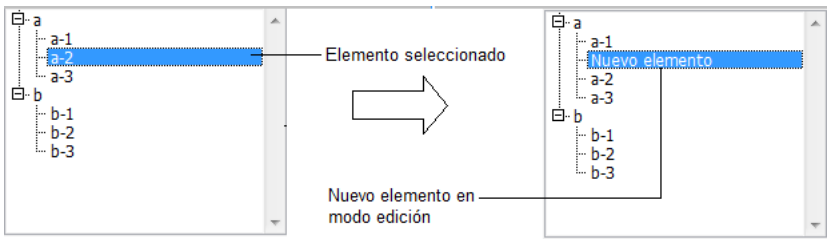
- En subformularios y formularios listados, el comando pasa a modo edición el primer campo de la fila especificada, en el orden de entrada.
- En listboxes mostrados en modo jerárquico, si el elemento objetivo pertenece a un nivel jerárquico colapsado, este nivel (como también los posibles niveles padres) se desplegarán automáticamente para que la línea sea visible.

Ejemplo 1

Este comando puede ser particularmente útil cuando crea un nuevo elemento en una lista jerárquica. Cuando se llama el comando, el último elemento añadido o insertado en la lista se convierte automáticamente en editable, sin que el usuario tenga que efectuar alguna acción especifica.

El siguiente código puede ser el método de un botón que le permite insertar un nuevo elemento en una lista existente. El texto por defecto "Nuevo_elemento" está listo automáticamente para ser cambiado:

```
vlUniqueRef:=vlUniqueRef+1
INSERT LIST ITEM(hList;*;"Nuevo_elemento";vlUniqueRef)
EDIT ITEM(*;"MiLista")
```

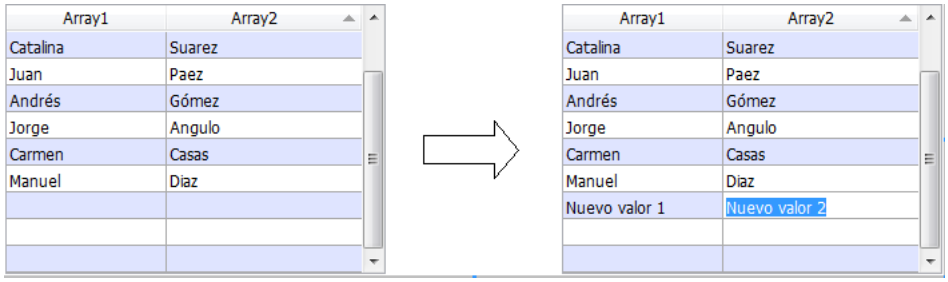


Ejemplo 2

Dadas dos columnas de un list box donde los nombres de las variables asociadas son respectivamente "Array1" y "Array2". El siguiente ejemplo inserta un nuevo elemento en dos arrays y pasa el nuevo elemento de Array2 a modo edición:

```
$vlRowNum:=Size of array(Array1)+1
LISTBOX INSERT ROWS(*;"MyListBox";$vlRowNum)
Array1{$vlRowNum}:="Nuevo valor 1"
```

```
Array2{$vlRowNum}:="Nuevo valor 2"
EDIT ITEM(Array2;$vlRowNum)
```



Array1	Array2
Catalina	Suarez
Juan	Paez
Andrés	Gómez
Jorge	Angulo
Carmen	Casas
Manuel	Diaz

Array1	Array2
Catalina	Suarez
Juan	Paez
Andrés	Gómez
Jorge	Angulo
Carmen	Casas
Manuel	Diaz
Nuevo valor 1	Nuevo valor 2

FILTER KEYSTROKE

FILTER KEYSTROKE (carFiltro)

Parámetro	Tipo		Descripción
carFiltro	Cadena	➡	Carácter de filtrado de tecleo o Cadena vacía para cancelar el filtrado

Descripción

FILTER KEYSTROKE permite reemplazar el carácter introducido por el usuario en un campo o en un área editable por el primer carácter de la cadena *carFiltro*.

Si pasa una cadena vacía, el filtrado se cancela y se ignora.

Generalmente, **FILTER KEYSTROKE** se llama en un método de formulario o de objeto cuando se gestiona el evento de formulario **On Before Keystroke**. Para detectar los eventos de tecleo, utilice el comando **Form event**. Para obtener el carácter tecleado, utilice el comando **Keystroke**.

IMPORTANTE: el comando **FILTER KEYSTROKE** le permite cancelar o reemplazar el carácter introducido por el usuario con otro carácter. Por otra parte, si quiere insertar más de un carácter para un keystroke especifico, recuerde que el texto que ve en la pantalla aún NO es el valor de la variable o campo fuente de datos para el área que está siendo editada. El valor del campo o de la variable fuente de datos se asigna después de que se valida la entrada de datos para el área. Por lo tanto depende de usted el colocar los datos introducidos en una variable y luego trabajar con el valor de la variable y reasignar el área de entrada (ver el ejemplo en esta sección).

Utilice el comando **FILTER KEYSTROKE** para:

- efectuar un filtro personalizado de caracteres
- crear un filtro de entrada no disponible en estándar, por ejemplo en los filtros de entrada
- implementar áreas dinámicas de búsqueda o de tecleo anticipado

Advertencia: si llama al comando **Keystroke** después de llamar a **FILTER KEYSTROKE**, se devuelve el carácter que usted pasa a este comando en lugar del carácter introducido realmente.

Ejemplo 1

Utilizando el siguiente código:

```
` Método de objeto del área editable miObjeto
Case of
   :(Form event=On Load)
      miObjeto:=""
   :(Form event=On Before Keystroke)
      If(Position(Keystroke;"0123456789")>0)
         FILTER KEYSTROKE("*")
      End if
End case
```

Todos los dígitos introducidos en el área *miObjeto* se transforman en asteriscos.

Ejemplo 2

Este código define el comportamiento de un área de entrada de contraseña, en la cual los caracteres introducidos son reemplazados (en la pantalla) por caracteres aleatorios:

```
` Método de objeto del área editable vsContraseña
Case of
   :(Form event=On Load)
      vsContraseña:=""
      vsContraseñaReal:=""
   :(Form event=On Before Keystroke)
      Manejo keystroke(->vsContraseña;->vsContraseñaReal)
      If(Position(Keystroke;Char(Backspace)+Char(Left arrow key)+
         Char(Right arrow key)+Char(Up arrow key)+Char(Down arrow key))=0)
         FILTER KEYSTROKE(Char(65+(Random%26)))
      End if
End case
```

Una vez validada la entrada de datos, usted recupera la contraseña introducida en realidad por el usuario *vsContraseñaReal*.
Nota: El método **Manejo keystroke** está listado en el ejemplo del comando **Keystroke.**

Ejemplo 3

En su aplicación, tiene algunas áreas de texto en las cuales puede introducir algunas frases. Su aplicación también incluye una tabla de diccionario de los términos utilizados con más frecuencia en su base. Durante la edición de sus áreas de texto, a usted le gustaría poder recuperar e insertar rápidamente entradas del diccionario basado en los caracteres seleccionados en un área de texto. Hay dos formas de hacer esto:
- Ofrecer algunos botones con teclas asociadas o
- Interceptar caracteres especiales durante la edición del área de texto

Este ejemplo implementa la segunda solución, basado en la tecla Ayuda.

Como se explicó anteriormente, durante la edición del área de texto, el valor introducido será asignará a la fuente de datos para esta área después de validar la entrada de datos. Para poder recuperar e insertar entradas del diccionario en el área de texto mientras se edita esta área, debe crear una segunda área para poner los valores introducidos. Se pasan como primeros parámetros los punteros hacia el área de entrada y hacia la variable, luego como tercer parámetro la cadena de caracteres "prohibidos". Sin importar cómo se trate el tecleado, el método devuelve el tecleadooriginal. Los caracteres "prohibidos" son aquellos que usted no quiere insertar en el área editable y quiere tratar como caracteres especiales.

```
  ` Método de proyecto Tecleado sombra
  ` Tecleado sombra ( Puntero ; Puntero ; Alfa) -> Alfa
  ` Tecleado sombra ( -> srcArea ; -> curValor ; Filtro ) -> Antiguo valor tecleado
C_STRING(1;$0)
C_POINTER($1;$2)
C_TEXT($vtNuevoValor)
C_STRING(255;$3)
  ` Devuelve el carácter original
$0:=Keystroke
  ` Obtener la selección de texto en el área editable
GET HIGHLIGHT($1->;$vlInicio;$vlFin)
  `Comenzar a trabajar con el valor actual
$vtNuevoValor:=$2->
  ` Dependiendo de la tecla presionada o del carácter introducido,
  ` Efectuar las acciones apropiadas
Case of
  ` La tecla Retorno (Suprimir) ha sido presionada
  :(Character code($0)=Backspace)
  ` Suprimir los caracteres seleccionados o el carácter a la izquierda del cursor
     $vtNuevoValor:=Borrar texto($vtNuevoValor;$vlInicio;$vlFin)
  ` Una tecla flecha ha sido presionada
  ` No hacer nada, sino aceptar el carácter tecleado
  :(Character code($0)=Left arrow key)
  :(Character code($0)=Right arrow key)
  :(Character code($0)=Up arrow key)
  :(Character code($0)=Down arrow key)

  ` Un carácter válido ha sido introducido
  :(Position($0;$3)=0)
     $vtNuevoValor:=Insertar texto($vtNuevoValor;$vlInicio;$vlFin;$0)
  Else
  ` El carácter no es aceptado
     FILTER KEYSTROKE("")
End case
  ` Devolver el valor para la próxima gestión de keystroke
$2->:=$vtNuevoValor
```

Este método utiliza los siguientes dos submétodos:

```
  ` Método de proyecto Borrar texto
  ` Suprimir texto ( Alfa; Long ; Long ) -> Alfa
  ` Suprimir texto ( -> Texto ; SelInicio ; SelFin ) -> Nuevo texto
C_TEXT($0;$1)
C_LONGINT($2;$3)
$0:=Substring($1;1;$2-1-Num($2=$3))+Substring($1;$3)
```

```
  ` Método de proyecto Insertar texto
  ` Insertar texto ( Alfa ; Long ; Long ; Alfa) -> Alfa
  ` Insertar texto ( -> srcText ; SelInicio ; SelFin ; Texto a insertar ) -> Nuevo texto
C_TEXT($0;$1;$4)
C_LONGINT($2;$3)
$0:=$1
If($2#$3)
   $0:=Substring($0;1;$2-1)+$4+Substring($0;$3)
Else
   Case of
      :($2<=1)
         $0:=$4+$0
```

```
        :($2>Length($0))
            $0:=$0+$4
        Else
            $0:=Substring($0;1;$2-1)+$4+Substring($0;$2)
    End case
End if
```

Una vez haya añadido estos métodos de proyecto a su base, puede utilizarlos de esta manera:

```
  ` Método de objeto del área editable vsDescripcion
Case of
    :(Form event=On Load)
        vsDescripcion:=""
        vsShadowDescripcion:=""
    ` Establecer la lista de caracteres "prohibidos" a tratar como teclas especiales
    ` ( acá, en este ejemplo, sólo la tecla Help es filtrada)
        vsSpecialKeys:=Char(HelpKey)
    :(Form event=On Before Keystroke)
        $vsKey:=Tecleado sombra(->vsDescripcion;->vsShadowDescripcion;vsSpecialKeys)
        Case of
            :(Character code($vsKey)=Help key)
    ` Hacer algo cuando la tecla Ayuda sea presionada
    ` Acá, en este ejemplo, una entrada de diccionario debe ser buscada e insertada
                CONSULTAR DICCIONARIO(->vsDescripcion;->vsShadowDescripcion)
        End case
End case
```

El método de proyecto **LOOKUP DICTIONARY** es listado a continuación. Su propósito es utilizar la variable shadow para reasignar el área editable a modificar:

```
  ` Método de proyecto CONSULTAR DICCIONARIO
  ` CONSULTAR DICCIONARIO ( Puntero ; Puntero )
  ` CONSULTAR DICCIONARIO ( -> Area editable ; ->ShadowVariable )

C_POINTER($1;$2)
C_LONGINT($vlInicio;$vlFin)

  ` Obtener la selección de texto en el área editable
GET HIGHLIGHT($1->;$vlInicio;$vlFin)
  ` Obtener el texto seleccionado o la palabra situada a la izquierda del cursor
$vtHighlightedText:=ObtenerTextoSeleccionado($2->;$vlInicio;$vlFin)
  `¿Hay algo que buscar?
If($vtHighlightedText#"")
  ` Si la selección de texto era el cursor
  ` la selección comienza con la palabra situada después del cursor
    If($vlInicio=$vlFin)
        $vlInicio:=$vlInicio-Length($vtHighlightedText)
    End if
  ` Buscar la primera entrada disponible del diccionario
    QUERY([Diccionario];[Diccionario]Entry=$vtHighlightedText+"@")
  ` ¿Hay alguna?
    If(Records in selection([Diccionario])>0)
  ` Si hay alguna entrada disponible, insertarla en el texto shadow
        $2->:=Insert text($2->;$vlInicio;$vlFin;[Diccionario]Entry)
  ` Copiar el texto shadow en área editable
        $1->:=$2->
  ` Fijar la selección justo después de insertar la entrada del diccionario
        $vlFin:=$vlInicio+Length([Diccionario]Entry)
        HIGHLIGHT TEXT(vsComments;$vlFin;$vlFin)
    Else
  ` No hay una entrada correspondiente en el diccionario
        BEEP
    End if
Else
  ` No hay un texto seleccionado
    BEEP
End if
```

El método **ObtenerTextoSeleccionado** es el siguiente:

```
  ` Método de objeto ObtenerTextoSeleccionado
  ` ObtenerTextoSeleccionado( Alfa ; Long ; Long ) -> Alfa
```

```
` ObtenerTextoSeleccionado ( Text ; SelInicio ; SelEnd ) -> texto seleccionado
C_TEXT($0;$1)
C_LONGINT($2;$3)
If($2<$3)
   $0:=Substring($1;$2;$3-$2)
Else
   $0:=""
   $2:=$2-1
   Repeat
      If($2>0)
         If(Position($1[[$2]];"  ,.!?:;()-_–—")=0)
            $0:=$1[[$2]]+$0
            $2:=$2-1
         Else
            $2:=0
         End if
      End if
   Until($2=0)
End if
```

Get edited text

Get edited text -> Resultado

Parámetro	Tipo	Descripción
Resultado	Texto	Texto en proceso de introducción

Descripción

El comando **Get edited text** se utiliza principalmente con el evento formulario On After Keystroke para recuperar el texto a medida que es introducido. También puede utilizarse con el evento formulario On Before Keystroke. Para mayor información sobre estos eventos formulario, por favor consulte la descripción del comando **Evento formulario** .

La combinación de este comando con los eventos formulario On Before Keystroke y On After Keystroke funciona de la siguiente manera:

- Tan pronto como un carácter se escribe en el teclado, se genera el evento On Before Keystroke. En este caso, la función **Get edited text** devuelve el contenido del área antes de que ocurriera la última pulsación de tecla. Por ejemplo, si el área contiene "PA" y el usuario digita una "R", **Get edited text** devuelve "PA" en el evento On Before Keystroke. Si el área está vacía inicialmente, **Get edited text** rdevuelve una cadena vacía.
- A continuación, se genera el evento formulario On After Keystroke. En este caso, **Get edited text** devuelve el contenido del área, incluyendo el último carácter introducido en el teclado. Por ejemplo, cuando el área contiene "PA" y el usuario digita una "R", **Get edited text** devuelve "PAR" en el evento On After Keystroke.

Estos dos eventos sólo se generan en los métodos objeto en cuestión.

Cuando se utiliza en un contexto diferente a la entrada de datos en un objeto de formulario, esta función devuelve una cadena vacía.

Ejemplo 1

El siguiente método convierte automáticamente los caracteres introducidos en mayúsculas:

```
If(Form event=On After Keystroke)
   [Viajes]Agencias:=Uppercase(Get edited text)
End if
```

Ejemplo 2

Este es un ejemplo de cómo procesar inmediatamente los caracteres introducidos en un campo tipo texto. La idea consiste en ubicar en otro campo texto (llamado "Palabras") todas las palabras de la frase que están siendo escritas. Para hacerlo, escriba el siguiente código en el método de objeto del campo:

```
If(Form event=On After Keystroke)
   $EntradaTiempoReal:=Get edited text
   PLATFORM PROPERTIES($platform)
   If($platform#3)  ` Mac OS
      Repeat
         $FraseDescompuesta:=Replace string($EntradaTiempoReal;Char(32);Char(13))
      Until(Position(" ";$FraseDescompuesta)=0)
   Else  ` Windows
      Repeat
         $FraseDescompuesta:=Replace string($EntradaTiempoReal;Char(32);Char(13)+Char(10))
      Until(Position(" ";$FraseDescompuesta)=0)
   End if
   [Ejemplo]Palabras:=$FraseDescompuesta
End if
```

Nota: este ejemplo no es exhaustivo porque hemos asumido que las palabras se separan únicamente por espacios (Char (32)). Para una solución completa necesitará añadir otros filtros para extraer todas las palabras (comas, punto y comas, apóstrofes, etc.).

GET HIGHLIGHT

GET HIGHLIGHT ({* ;} objeto ; inicioSel ; finSel)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Campo, Variable, Objeto de formulario	➡	Nombre del objeto (si se especifica *) o campo o variable (si se omite *)
inicioSel	Entero largo	⬅	Posición del inicio de la selección de texto
finSel	Entero largo	⬅	Posición del fin de la selección de texto

Descripción

El comando **GET HIGHLIGHT** permite determinar el texto seleccionado actualmente en *objeto*.

Si pasa el parámetro opcional *, indica que el parámetro *objeto* es un nombre de objeto (cadena). Si no pasa el parámetro *, indica que el parámetro *objeto* es un campo o variable. En este caso, pase la referencia del campo o variable (campos o variables de formulario únicamente) en lugar de una cadena.

Nota: este comando no puede utilizarse con campos ubicados en el formulario listado de un subformulario.

El texto puede ser seleccionado por el usuario o por el comando **HIGHLIGHT TEXT**.

El parámetro *inicioSel* devuelve la posición del primer carácter seleccionado.
El parámetro *finSel* devuelve la posición del último carácter seleccionado más uno.

Si los valores devueltos de *inicioSel* y *finSel* son iguales, el usuario no ha seleccionado ningún texto, y el punto de inserción está ubicado antes del carácter especificado por *inicioSel*.

Si el objeto designado por el parámetro *objeto* no se encuentra en el formulario, el comando devuelve -1 en *inicioSel* y -2 en *finSel*.

Ejemplo 1

El siguiente ejemplo obtiene el texto seleccionado en el campo *[Productos]Comentarios*:

```
GET HIGHLIGHT([Productos]Comentarios;vFirst;vLast)
If(vFirst<vLast)
   ALERT("El texto seleccionado es: "+Substring([Productos]Comentarios;vPrimerot;vUltimo-vPrimero))
End if
```

Ejemplo 2

Ver el ejemplo del comando **FILTER KEYSTROKE**.

Ejemplo 3

Modificación del estilo del texto resaltado:

```
GET HIGHLIGHT(*;"miTexto";$startsel,$endsel)
ST SET ATTRIBUTES(*;"miTexto";$startsel,$endsel;Attribute underline style;1;Attribute bold style;1)
```

GOTO OBJECT

GOTO OBJECT ({* ;} objeto)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica = objeto es un nombre de objeto (cadena) Si se omite = objeto es un campo o una variable
objeto	Campo, Variable	⇒	Nombre del objeto (si se especifica *) o Campo o Variable (si se omite *) a donde ir

Descripción

El comando **GOTO OBJECT** se utiliza para seleccionar el objeto editable *objeto* como el área activa del formulario. Es equivalente de un clic del usuario en el área o de utilizar la tecla Tab para seleccionar el campo o la variable.

Si especifica el parámetro opcional *, indica un nombre de objeto (una cadena) en *objeto*. Si omite el parámetro opcional *, indica un campo o una variable en *objeto*. En este caso, especifique una referencia de campo o de variable (objetos de campos o variables únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección .

Para eliminar todo foco en el formulario actual, llame al comando mientras pasa un nombre de objeto vacío en *objeto* (ver ejemplo 2).

El comando **GOTO OBJECT** puede utilizarse en el contexto de un subformulario. Cuando se llama desde un subformulario, busca primero el objeto en el subformulario, luego, si la búsqueda no encuentra nada allí, extiende la búsqueda a objetos del formulario padre.

Ejemplo 1

El comando **GOTO OBJECT** puede utilizarse de dos maneras:

```
GOTO OBJECT([Personas]Nombre) ` Referencia del campo
GOTO OBJECT(*;"AreaEdad") ` Nombre del objeto
```

Ejemplo 2

Si no quiere que ningún objeto del formulario tenga el foco.

```
GOTO OBJECT(*;"")
```

Ejemplo 3

Ver el ejemplo del comando **REJECT**.

HIGHLIGHT TEXT

HIGHLIGHT TEXT ({* ;} objeto ; inicioSel ; finSel)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena)Si se omite, objeto es un campo o una variable
objeto	Campo, Variable, Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Campo o variable (si se omite *)
inicioSel	Entero largo	➡	Nueva posición de inicio de selección de texto
finSel	Entero largo	➡	Nueva posición de fin de selección de texto

Descripción

El comando **HIGHLIGHT TEXT** selecciona una parte de texto en *objeto*.

Si pasa el parámetro opcional *, indica que el parámetro *objeto* es un nombre de un objeto (una cadena) Si no pasa el parámetro *, indica que el parámetro *objeto* es un campo o una variable. En este caso, pase la referencia del campo o de la variable (campos o variables de formulario únicamente) en lugar de una cadena.

Si *objeto* no es el objeto que está siendo modificado, esta área recupera el foco.

Nota: este comando no puede utilizarse con campos en un subformulario.

El parámetro *inicioSel* representa la posición del primer carácter a seleccionar, y el parámetro *finSel* representa la posición del último carácter a seleccionar más uno. Si *inicioSel* y *finSel* son iguales, el punto de inserción está ubicado antes del carácter especificado por *inicioSel*, y ningún carácter está seleccionado.

Si *finSel* es superior al número de caracteres en *objeto*, todos los caracteres entre *inicioSel* y el final del texto son seleccionados.

Ejemplo 1

El siguiente ejemplo selecciona todos los caracteres en el campo editable *[Productos]Notas*:

```
HIGHLIGHT TEXT([Productos]Notas;1;Length([Productos]Notas)+1)
```

Ejemplo 2

El siguiente ejemplo mueve el punto de inserción al principio del campo editable *[Productos]Notas*:

```
HIGHLIGHT TEXT([Productos]Notas;1;1)
```

Ejemplo 3

El siguiente ejemplo mueve el punto de inserción al final del campo editable *[Productos]Notas*:

```
$vLen:=Length([Productos]Notas)+1HIGHLIGHT TEXT([Productos]Notas;$vLen;$vLen)
```

Ejemplo 4

Ver el ejemplo del comando **FILTER KEYSTROKE**.

Keystroke

Keystroke -> Resultado

Parámetro	Tipo	Descripción
Resultado	Cadena	Carácter introducido por el usuario

Descripción

Keystroke devuelve el carácter introducido por el usuario en un campo o en un área editable.

Generalmente, **Keystroke** se llama en un método de formulario o de objeto durante la gestión del evento de formulario **On Before Keystroke**. Para detectar eventos de tecleo, utilice el comando **Form event**.

Para reemplazar el carácter introducido por el usuario con otro carácter, utilice el comando **FILTER KEYSTROKE**.

Nota: la función **Keystroke** no funciona en subformularios.

IMPORTANTE: si quiere efectuar algunas operaciones "al vuelo" dependiendo del valor actual del área de entrada que está siendo editada, así como del nuevo carácter a introducir, recuerde que el texto que ve en la pantalla NO ES aún el valor del campo o de la variable fuente. El valor del campo o de la variable fuente de datos se asigna después de que se valida la entrada de datos para el área (tabulación en otra área, clic en un botón, etc). Por lo tanto depende de usted el colocar los datos introducidos en una variable y luego trabajar con el valor de la variable. Debe hacer esto si necesita saber el valor actual del texto para efectuar acciones especiales. Igualmente puede utilizar la función **Get edited text**.

Puede utilizar el comando **Keystroke** para:

- efectuar un filtro personalizado de caracteres
- crear un filtro de entrada no disponible en estándar, por ejemplo en los filtros de entrada
- implementar áreas dinámicas de búsqueda o de tecleo anticipado

Ejemplo 1

Consulte los ejemplos del comando **FILTER KEYSTROKE**.

Ejemplo 2

Cuando procesa un evento **On Before Keystroke**, usted está administrando la edición del área de texto actual (donde se encuentra el cursor), no el "valor futuro" de la fuente de datos (campo o variable) de esta área. El método de proyecto Manejo tecleo permite colocar en una segunda variable los datos introducidos en un área de texto, de manera que usted puede utilizar esta variable para efectuar diferentes acciones mientras introduce caracteres en el área. Usted pasa como primer parámetro un puntero hacia la fuente de datos del área, y como segundo parámetro un puntero hacia la segunda variable. El método devuelve el nuevo valor del área de texto en la segunda variable, y devuelve **True** si este valor es diferente del valor antes de la entrada del último carácter.

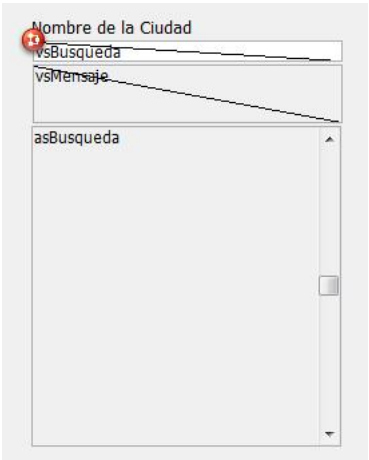
```
  ` Método de proyecto Manejo tecleo
  ` Manejo tecleo ( Puntero ; Puntero ) -> Booleano
  ` Manejo tecleo ( -> srcArea ; -> curValor ) -> Es un nuevo valor

C_POINTER($1;$2)
C_TEXT($vtNuevoValor)

  ` Obtener el texto seleccionado en el área editable
GET HIGHLIGHT($1->;$vlInicio;$vlFin)
  ` Comenzar a trabajar con el valor actual
$vtNuevoValor:=$2->
  ` Dependiendo de la tecla presionada o del carácter introducido,
  ` Realizar las acciones apropriadas
Case of

  ` La tecla Retroceso ha sido presionada
  :(Character code(Keystroke)=Backspace)
  ` Suprimir los caracteres seleccionados o el carácter a la izquierda del cursor
     $vtNuevoValor:=Substring($vtNuevoValor;1;$vlInicio-1-Num($vlInicio=$vlFin))
     +Substring($vtNuevoValor;$vlFin)

  ` Un carácter aceptable ha sido introducido
  :(Position(Keystroke;"abcdefghjiklmnopqrstuvwxyz -0123456789")>0)
     If($vlInicio#$vlFin)
  ` Uno o varios caracteres son seleccionados, el keystroke va a borrarlos
        $vtNuevoValor:=Substring($vtNuevoValor;1;$vlInicio-1)
        +Keystroke+Substring($vtNuevoValor;$vlFin)
     Else
  ` La selección de texto es el cursor
```

```
            Case of
   ` El cursor está actualmente al comienzo del texto
               :($vlInicio<=1)
   ` Inserción del carácter al principio del texto
                  $vtNuevoValor:=Keystroke+$vtNuevoValor
   ` El cursor está actualmente al final del texto
               :($vlInicio>=Length($vtNuevoValor))
   ` Añadir el carácter al final del texto
                  $vtNuevoValor:=$vtNuevoValor+Keystroke
               Else
   ` El cursor se encuentra en el texto, insertar el nuevo carácter
                  $vtNuevoValor:=Substring($vtNuevoValor;1;$vlInicio-1)+Keystroke
                  +Substring($vtNuevoValor;$vlInicio)
            End case
         End if

   ` Una tecla flecha ha sido presionada
   ` No haga nada, sólo acepte el carácter tecleado
     :(Character code(Keystroke)=Left arrow key)
     :(Character code(Keystroke)=Right arrow key)
     :(Character code(Keystroke)=Up arrow key)
     :(Character code(Keystroke)=Down arrow key)
   `
     Else
   ` No acepte caracteres diferentes de letras, dígitos, espacios y guiones
         FILTER KEYSTROKE("")
End case
   ` ¿Es diferente el valor ahora?
$0:=($vtNuevoValor#$2->)
   ` Devolver el valor para la gestión del próximo keystroke
$2->:=$vtNuevoValor
```

Una vez este método de proyecto se añada a su aplicación, puede utilizarlo de la siguiente forma:

```
   ` Método de objeto del área de entrada MiObjeto
Case of
   :(Form event=On Load)
      MiObjeto:=""
      MiObjetoCaché:=""
   :(Form event=On Before Keystroke)
      If(Manejo tecleo(->MiObjeto;->MiObjetoCaché))
   ` Efectuar las acciones apropiadas utilizando el valor almacenado en MiObjetoCaché
      End if
End case
```

Examinemos por ejemplo el siguiente formulario:



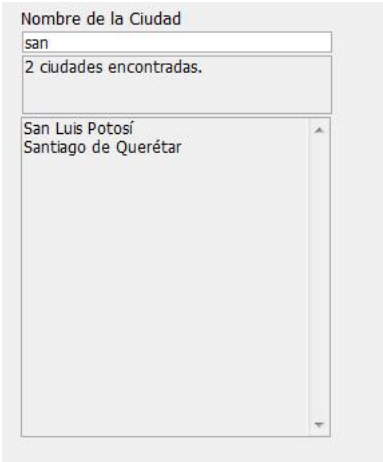
Esta compuesto de los siguientes objetos: un área editable *vsBusqueda*, un área no editable *vsMensaje*, y un área de desplazamiento *asBusqueda*. Durante la entrada de caracteres en *vsBusqueda*, el método para ese objeto efectúa una búsqueda en la tabla [Codigos postales], permitiendo al usuario encontrar ciudades solamente presionando los primeros caracteres de los nombres de la ciudades.

Este es el método de objeto *vsBusqueda*:

```
   ` Método de objeto del área de entrada vsBusqueda
Case of
   :(Form event=On Load)
      vsBusqueda:=""
```

```
        vsResult:=""
        vsMensaje:="Introduzca los primeros caracteres de la ciudad que busca."
        CLEAR VARIABLE(asBusqueda)
    :(Form event=On Before Keystroke)
        If(Manejo tecleo(->vsBusqueda;->vsResult))
            If(vsResult#"")
                QUERY([Codigos postales];[Codigos postales]Ciudad=vsResult+"@")
                MESSAGES OFF
                DISTINCT VALUES([Codigos postales]Ciudad;asBusqueda)
                MESSAGES ON
                $vlResult:=Size of array(asBusqueda)
                Case of
                    :($vlResult=0)
                        vsMensaje:="No se encontró ninguna ciudad."
                    :($vlResult=1)
                        vsMensaje:="Se encontró una ciudad."
                    Else
                        vsMensaje:=String($vlResult)+" ciudades encontradas."
                End case
            Else
                DELETE FROM ARRAY(asBusqueda;1;Size of array(asBusqueda))
                vsMensaje:="Introduzca los primeros caracteres de la ciudad que está buscando."
            End if
        End if
End case
```

Este es el formulario en ejecución:



Utilizando las habilidades de la comunicación interproceso de 4D, puede construir interfaces de usuario en las cuales las características de búsqueda se ofrezcan en ventanas flotantes que se comuniquen con procesos en los cuales los registros son listados o editados.

Corrector ortográfico

- SPELL ADD TO USER DICTIONARY
- SPELL CHECK TEXT
- SPELL CHECKING
- SPELL Get current dictionary
- SPELL GET DICTIONARY LIST
- SPELL SET CURRENT DICTIONARY

SPELL ADD TO USER DICTIONARY

SPELL ADD TO USER DICTIONARY (palabras)

Parámetro	Tipo		Descripción
palabras	Texto, Array texto	➡	Palabra o lista de palabras para agregar al diccionario del usuario

Descripción

El comando **SPELL ADD TO USER DICTIONARY** añade una o más palabras al diccionario usuario actual.

El diccionario usuario es un dicccionario que contiene palabras añadidas por el usuario al diccionario actual. Este diccionario es un archivo llamado *UserDictionaryxxx.dic* (donde *xxx* representa el ID del diccionario actual) que se crea automáticamente en la carpeta 4D actual. Hay un diccionario usuario por cada diccionario actual utilizado.

Puede pasar en *palabras* una cadena texto o un array texto con las palabras a añadir al diccionario usuario. Si una de las palabras ya está en el diccionario, es ignorada por el comando.

Ejemplo

Adición de nombres propios al diccionario de usuario:

```
ARRAY TEXT($arrTwords;0)
APPEND TO ARRAY($arrTwords;"4D")
APPEND TO ARRAY($arrTwords;"Wakanda")
APPEND TO ARRAY($arrTwords;"Clichy")
SPELL ADD TO USER DICTIONARY($arrTwords)
```

SPELL CHECK TEXT

SPELL CHECK TEXT (texto ; posErr ; longErr ; posVerif ; arrSug)

Parámetro	Tipo		Descripción
texto	Texto	➡	Texto a verificar
posErr	Entero largo	⬅	Posición del primer carácter de la palabra desconocida
longErr	Entero largo	⬅	Longitud de la palabra desconocida
posVerif	Entero largo	➡	Posición de inicio de la verificación
arrSug	Array texto	⬅	Lista de sugerencias

Descripción

El comando **SPELL CHECK TEXT** verifica el contenido del parámetro *texto* a partir del carácter *posVerif* y devuelve la posición de la primera palabra desconocida encontrada (si la hay).

Este comando devuelve la posición del primer carácter de esta palabra desconocida en *posErr* y su longitud en *longErr*. El array *arrSug* recibe la(s) sugerencia(s) de corrección propuestas por el corrector ortográfico.

Si la verificación inicia sin error y se encuentra una palabra desconocida, la variable sistema OK toma el valor 0. Si un error de inicialización ocurre durante la verificación o si no se encuentran palabras desconocidas, OK toma el valor 1.

Note OS X: bajo OS X, cuando el corrector nativo se activa, este comando no soporta la corrección gramatical.

Ejemplo

Queremos contar el número posible de errores en un texto:

```
$pos:=1
$errCount:=0
ARRAY TEXT($tErrors;0)
ARRAY TEXT($tSuggestions;0)
Repeat
   SPELL CHECK TEXT($myText;$errPos;$errLength;$pos;$tSuggestions)
   If(OK=0)
      $errCount:=$errCount+1  // contador de errores
      $errorWord:=Substring($myText;$errPos;$errLength)
      APPEND TO ARRAY($errors;$errorWord)  // array de errores
      $pos:=$errPos+$errLength   //continuar la verificación
   End if
Until(OK=1)
 // Al final $errCount=Size of array($errorWord)
```

SPELL CHECKING

SPELL CHECKING
Este comando no requiere parámetros

Descripción

El comando **SPELL CHECKING** activa la revisión ortográfica del campo o variable que tiene el foco en el formulario en pantalla. El objeto verificado debe ser de tipo Alfa o Texto.

Nota: si desea activar el corrector ortográfico haciendo clic en un botón en el formulario, asegúrese de que este botón no tenga la propiedad "enfocable".

La verificación ortográfica comienza con la primera palabra del campo o variable. Si se detecta una palabra desconocida, aparece la caja de diálogo de corrección (para mayor información, consulte el Manual de Diseño de 4D). 4D utiliza el diccionario actual (correspondiente al lenguaje de la aplicación) a menos que haya utilizado el comando **SPELL SET CURRENT DICTIONARY**.

Atención: el comando **SPELL CHECKING** afecta el texto que se está introduciendo en el formulario, y no la fuente de datos asociada (campo o variable). Esto significa que si llama a este comando desde los eventos de formulario On Data Change o On Losing Focus (no recomendado), esto no afectará el texto almacenado ya que 4D ya ha asignado el texto introducido a la fuente de datos. En este caso, es necesario asignar el resultado editado a la fuente de datos, usando el comando **Get edited text**. Por ejemplo:

```
If(Form event=On Data Change)
   SPELL CHECKING
   theVariable:=Get edited text
End if
```

SPELL Get current dictionary

SPELL Get current dictionary -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		ID del diccionario utilizado para la corrección ortográfica

Descripción

El comando **SPELL Get current dictionary** devuelve el número de ID del diccionario que está siendo utilizado.

Ejemplo

Queremos mostrar el lenguaje del diccionario actual:

```
 // Lista de los diccionarios cargados
SPELL GET DICTIONARY LIST($IDs_al;$Codes_at;$Names_at)
$curLangCode:=SPELL Get current dictionary
$countryName:=$Names_at{Find in array($IDs_al;$curLangCode)}
 // Mostrar mensaje
ALERT("Diccionario actual: "+$countryName)  // Español
```

SPELL GET DICTIONARY LIST

SPELL GET DICTIONARY LIST (lengID ; lengArchivos ; lengNoms)

Parámetro	Tipo		Descripción
lengID	Array entero largo	⇐	IDs únicos de los lenguajes
lengArchivos	Array texto	⇐	Nombres de los archivos de lenguaje instalados
lengNoms	Array texto	⇐	Nombres locales de los lenguajes

Descripción

El comando **SPELL GET DICTIONARY LIST** devuelve en los arrays *lengID*, *lengArchivos* y *lengNoms*, los IDs, los nombres de archivos y los nombres de lenguajes correspondientes a los archivos de diccionarios Hunspell instalados en el equipo.

Nota: para mayor información sobre los diccionarios Hunspell, consulte la sección **Corrección ortográfica** en el *Manual de Diseño*.

- *lengID* recibe los números de ID generados automáticamente y utilizados con el comando **SPELL SET CURRENT DICTIONARY**.
  Note que los IDs son únicos y basados en los nombres de los archivos. Este comando es útil principalmente en la fase de desarrollo; no tiene que regenerar los IDs cada vez que la base se ejecuta.
- *lengArchivos* recibe los nombres de los archivos de diccionarios instalados en la máquina.
- *lengNoms* recibe los nombres de los lenguajes expresados en el lenguaje actual de la aplicación. Por ejemplo, para un diccionario francés, el valor "français (France)" será devuelto en una máquina configurada en francés y "French (France)" en un sistema inglés. El nombre del lenguaje es seguido por "- Hunspell". Este campo sólo es válido para los archivos "conocidos" por 4D. Para los archivos no conocidos (por ejemplo, archivos personalizados), se devuelve el nombre "N/A - Hunspell". Este principio no le impide utilizar el diccionario (si el archivo correspondiente es válido), el ID devuelto puede ser pasado al comando **SPELL SET CURRENT DICTIONARY**.

Ejemplo

Usted pone "fr-classic+reform1990.aff" y "fr-classic+reform1990.dic" como también "fr-dentist.aff" y "fr-dentist.dic" en el directorio Hunspell:

```
ARRAY LONGINT($langID;0)
ARRAY TEXT($dicName;0)
ARRAY TEXT($langDesc;0)
SPELL GET DICTIONARY LIST($langID;$dictName;$langDesc)
```

$langID	$dictName	$langDesc
65536	en_GB	English (UK)
65792	en_US	English (USA)
131072	de_DE	German (Germany)
196608	es_ES	Spanish
262144	fr_FR	French (France)
589824	nb_NO	Norwegian Bokmal (Norway)
1074036166	fr-classic+reform1990	French (France) - Hunspell
1073901273	fr-dentist	No description - Hunspell

SPELL SET CURRENT DICTIONARY

SPELL SET CURRENT DICTIONARY (diccionario)

Parámetro	Tipo		Descripción
diccionario	Entero largo, Texto	➡	ID, Nombre o Código de lenguaje del diccionario a utilizar para la corrección ortográfica Si se omite = restablecer el diccionario por defecto

Descripción

El comando **SPELL SET CURRENT DICTIONARY** reemplaza el diccionario actual por el especificado en el parámetro *diccionario*. El diccionario actual se utiliza para la corrección ortográfica integrada de 4D (para mayor información, consulte el *Manual de Diseño*) como también de 4D Write Pro. La modificación del diccionario actual repercute inmediatamente en todos los procesos de la base para la sesión, así como también en las áreas 4D Write Pro.

4D utiliza el diccionario:

- bajo Windows, el diccionario Hunspell correspondiente al lenguaje de la aplicación,
- por defecto bajo macOS, el corrector ortográfico nativo.

Nota: bajo macOS, puede utilizar el diccionario Hunspell con la ayuda del comando **SET DATABASE PARAMETER**. Para mayor información, consulte la sección **Configuración del corrector** en el manual de Diseño.

Puede utilizar el parámetro *diccionario* para cambiar el diccionario. Puede pasar:

- un número de identificación de diccionario Hunspell (devuelto por el comando **SPELL GET DICTIONARY LIST**),
- un nombre de diccionario Hunspell (que corresponde al nombre de archivo del diccionario Hunspell, con o sin la extensión),
- un código de lenguaje BCP 47, ISO 639-1 o ISO 639-2. Por ejemplo, con el código de lenguaje BCP 47, "en-US" designa Inglés Americano y "en-GB" designa Inglés británico. Estos códigos son redirigidos internamente al diccionario actual correspondiente (Hunspell o nativo macOS).

Nota de compatibilidad: en versiones anteriores de 4D, se soportaron los diccionarios "Cordial". Por compatibilidad, aún es posible pasar un número de diccionario "Cordial" en el parámetro *diccionario* (valor o constante del tema "**Diccionarios**"). En este caso, sin embargo, el diccionario se redirige internamente a un diccionario Hunspell equivalente (o el diccionario nativo en OS X).

Variables y conjuntos del sistema

Si el *diccionario* se carga correctamente, la variable sistema OK toma el valor 1; de lo contrario, toma el valor 0 y devuelve un error.

Ejemplo

Carga del diccionario "fr-classic" presente en la carpeta Hunspell:

```
SPELL SET CURRENT DICTIONARY("fr-classic")
 // SPELL SET CURRENT DICTIONARY ("FR-classic.dic") es válido
```

Documentos del sistema

- Documentos del sistema
- Append document
- CLOSE DOCUMENT
- Convert path POSIX to system
- Convert path system to POSIX
- COPY DOCUMENT
- CREATE ALIAS
- Create document
- CREATE FOLDER
- DELETE DOCUMENT
- DELETE FOLDER
- DOCUMENT LIST
- Document to text
- FOLDER LIST
- GET DOCUMENT ICON
- Get document position
- GET DOCUMENT PROPERTIES
- Get document size
- Get localized document path
- MOVE DOCUMENT
- Object to path
- Open document
- Path to object
- RESOLVE ALIAS
- Select document
- Select folder
- SET DOCUMENT POSITION
- SET DOCUMENT PROPERTIES
- SET DOCUMENT SIZE
- SHOW ON DISK
- Test path name
- TEXT TO DOCUMENT
- VOLUME ATTRIBUTES
- VOLUME LIST
- *_o_Document creator*
- *_o_Document type*
- *_o_MAP FILE TYPES*
- *_o_SET DOCUMENT CREATOR*
- *_o_SET DOCUMENT TYPE*

Documentos del sistema

Introducción

Todos los documentos y aplicaciones que utiliza en su ordenador están almacenados en **archivos** en los discos duros **conectados** o **montados** en su ordenador, o en discos externos u otros dispositivos de almacenamiento. En 4D, utilizamos los términos **archivo** o **documento** para referirnos a estos documentos y aplicaciones. Sin embargo, la mayoría de los comandos en este tema utilizan el término "documento" porque generalmente los utilizará para acceder a los documentos (en lugar de aplicación o archivos sistema) en disco.

Un disco duro puede ser formateado de manera que tenga una o varias particiones, cada una de ellas llamada un **volumen**. No importa si dos volúmenes están físicamente presentes en el mismo disco duro; al nivel de 4D, generalmente estos volúmenes se consideran entidades separadas y equivalentes.

Un volumen puede estar ubicado en un disco duro conectado físicamente a su ordenador o montado en una red por medio de un protocolo de distribución de archivos tal como TCP/IP, AFP o SMB (Macintosh). Cualquiera que sea el caso, a nivel de 4D, estos volúmenes son considerados del mismo modo cuando utiliza los comandos del tema Documentos del sistema (excepto en casos especiales, por ejemplo al utilizar plug-ins para extender las capacidades de su aplicación en este dominio).

Cada volumen tiene un **nombre de volumen**. En Windows, los volúmenes están designados por una letra seguida por dos puntos. Generalmente **C:** y **D:** se utilizan para designar los volúmenes que usted utiliza para lanzar su sistema (a menos que usted configure su PC de otra forma). Luego las letras de la **E:** a la **Z:** se utilizan para volúmenes adicionales conectados o montados en su PC (drivers USB, drivers adicionales, drivers de red, etc.). En Macintosh, los volúmenes tienen nombres comunes; estos son los nombres que usted ve en el escritorio al nivel del Finder.

Generalmente, usted clasifica sus documentos en **carpetas**, que pueden contener otras carpetas. No es buena idea acumular cientos o miles de archivos en el mismo nivel de un volumen; es desordenado y vuelve lento su sistema. En Windows, una carpeta todavía se llama un **directorio**. Las carpetas siempre se han llamado de esta manera en Macintosh.

Para identificar un documento de manera única, necesita saber el nombre del volumen y el/los nombre(s) de la(s) carpeta(s) donde el documento está ubicado como también el nombre del documento mismo. Si concatena todos estos nombres, usted obtiene la **ruta de acceso** al documento. En el nombre esta ruta, los nombres de las carpetas están separados por un carácter especial llamado **separador de carpeta**. En Windows, este carácter es la barra oblicua inversa (**); en Macintosh son los dos puntos (*:*).

Veamos un ejemplo. Usted tiene un documento **Importante** ubicado en la carpeta **Memos**, que está en la carpeta **Documentos**, que está en la carpeta **Trabajo actual**.

En Windows, si todo está ubicado en el drive C: drive (volumen), la ruta de acceso al documento es:

C:\Trabajo actual\Documentos\Memos\Importante Memo.txt

Nota: igualmente el carácter \ es utilizado por el editor de métodos de 4D para designar las secuencias de escape. Para evitar todo problema de interpretación, el editor transforma automáticamente las rutas de acceso del tipo **C:\Disk** en **C:\\Disk**. Para mayor información, consulte el párrafo **Entrada de rutas de acceso Windows y secuencias de escape**.

En Macintosh, si todo el conjunto está ubicado en el disco (volumen) **Interno**, la ruta de acceso del documento es:

Interno:Trabajo actual:Documentos:Memos:Importante Memo.txt

Sin importar la plataforma, la ruta completa de un documento puede expresarse de esta manera:

VolNombre DirSep { DirNombre DirSep { DirNombre DirSep { ... } } } DocNombre

Todos los documentos (archivos) ubicados en los volúmenes tienen varias características, llamadas generalmente **atributos** o **propiedades**: por ejemplo el **nombre** del documento mismo y su **extensión**.

DocRef: número de referencia del documento

Un documento está **abierto en modo lectura/escritura**, **abierto en modo sólo lectura** o **cerrado**. Utilizando los comandos integrados de 4D, un documento puede abrirse en modo lectura/escritura por sólo un proceso a la vez. Un proceso puede abrir varios documentos, varios procesos pueden abrir múltiples documentos, usted puede abrir el mismo documento en modo sólo lectura tantas veces como sea necesario, pero no puede abrir dos veces simultáneamente el mismo documento en modo lectura/escritura.

Usted abre un documento con los comandos **Open document**, **Create document** y **Append document**. Los comandos **Create document** y **Append document** abren automáticamente los documentos en modo lectura/escritura. Sólo el comando **Open document** le permite elegir el modo de apertura. Una vez abierto un documento en lectura/escritura, usted puede leer y escribir caracteres en el documento (ver los comandos **RECEIVE PACKET** y **SEND PACKET**). Cuando termina con un documento, generalmente se cierra con el comando **CLOSE DOCUMENT**.

Se hace referencia a todos los documentos abiertos utilizando la expresión **DocRef** devuelta por los comandos **Open document**, **Create document** y **Append document**. DocRef identifica de manera única un documento abierto. Es una expresión de tipo Hora. Todos los comandos que funcionan con documentos abiertos esperan DocRef como parámetro. Si pasa un DocRef incorrecto a uno de estos comandos, se genera un error del administrador de archivos.

Nota: cuando se llama desde un proceso apropiativo, una referencia *DocRef* sólo se puede utilizar a partir de este proceso apropiativo. Cuando se llama desde un proceso cooperativo, una referencia *DocRef* se puede utilizar desde cualquier otro proceso cooperativo.

Gestión de errores E/S

Cuando accede a documentos (apertura, cierre, eliminación, cambio de nombre, copia), cuando cambia las propiedades de un documento o cuando lee y escribe caracteres en un documento, E/S pueden ocurrir errores. Un documento puede no encontrarse; puede estar bloqueado; puede ya estar abierto en modo escritura. Puede reparar estos errores con un método de

gestión de errores instalado por el comando **ON ERR CALL**. La mayoría de los errores que pueden ocurrir mientras se utilizan documentos del sistema se describen en la sección **Errores de gestión de archivos del SO**.

La variable sistema Document

Los comandos **Open document**, **Create document**, **Append document** y **Select document** permiten acceder a un documento utilizando las cajas de diálogo estándar de abrir o cerrar archivos. Cuando accede a un documento por medio de un diálogo estándar, 4D devuelve la ruta completa del documento en la variable sistema *Document*. Esta variable sistema tiene que distinguirse del parámetro *document* que aparece en la lista de parámetros de los comandos.

Se puede encontrar información adicional sobre la variable del sistema *Document* en la sección **Variables sistema**.

Entrada de rutas de acceso Windows y secuencias de escape

El editor de métodos de 4D permite utilizar secuencias de escape. Una secuencia de escape es un conjunto de caracteres que se utilizan para remplazar un carácter "especial". Le secuencia comienza con el carácter barra oblicua inversa \, seguido por un caracter. Por ejemplo, *\t* es la secuencia de escape para el carácter *Tab*.

El carácter \ también se utiliza como separador de rutas de acceso en Windows. Por lo general, 4D interpretará correctamente las rutas de acceso Windows que se introducen en el editor de métodos reemplazando automáticamente las barras simples \ con barras dobles \\. Por ejemplo, **C:\Carpeta** se convertirá en **C:\\Carpeta**.

Sin embargo, si escribe **C:\MisDocumentos\Nuevo**, 4D mostrará **C:\\MisDocumentos\Nuevo**. En este caso, el segundo \ es interpretado de manera incorrecta como *N* (una secuencia de escape existente). Por lo tanto debe introducir una barra doble \\ cuando quiera insertar una barra oblicua inversa delante de un carácter que se utiliza en una de las secuencias de escape reconocida por 4D.

Las siguientes son las secuencias de escape reconocidas por 4D:

Secuencia de escape	Caracter reemplazado
\n	LF (Nueva línea)
\t	HT (Tabulación)
\r	CR (Retorno de carro)
\\	\ (Barra oblicua inversa)
\"	" (Comillas)

Ruta de acceso absoluta o relativa

La mayoría de las rutinas de esta sección aceptan **nombres documento**, **rutas de acceso relativas** o **rutas de acceso absolutas**:

- Las **rutas de acceso relativas** definen una ubicación con respecto a una carpeta presente en el disco. Pasar sólo un **nombre de documento** se considera como utilizar una ruta de acceso relativa. En 4D, las rutas de acceso relativas se expresan por lo general respecto a la carpeta de la base, es decir a la carpeta que contiene el archivo de estructura.. Las rutas de acceso relativas son particularmente útiles para el despliegue de aplicaciones en entornos heterogéneos.
- Las **rutas de acceso absolutas** definen una ubicación a partir de la raíz de un volumen y no dependen de la posición actual de la carpeta de la base.

Para determinar si una ruta de acceso pasada a un comando debe interpretarse como absoluta o relativa, 4D aplica un algoritmo específico para cada plataforma.

Bajo Windows

Si el parámetro contiene únicamente los caracteres **y** si el segundo es un ':',
o si el texto contiene ':' y '\' como segundo y tercer carácter,
o si el texto comienza por "\\",
luego la ruta de acceso es **absoluta**.

En todos los demás casos, la ruta es **relativa**.

Ejemplos con el comando **CREATE FOLDER**:

```
CREATE FOLDER("lunes") // ruta relativa
CREATE FOLDER("\lunes") // ruta relativa
CREATE FOLDER("\lunes\martes") //ruta relativa
CREATE FOLDER("c:") // ruta absoluta
CREATE FOLDER("d:\lunes") // ruta absoluta
CREATE FOLDER("\\srv-Internal\temp") // ruta absoluta
```

Bajo Mac OS

Si el texto comienza con un separador de carpeta':',
o si no contiene ninguno,
luego la ruta de acceso es **relativa**.

En todos los demás casos, la ruta es **absoluta**.

Ejemplos con el comando **CREATE FOLDER**:

```
CREATE FOLDER("lunes") // ruta relativa
CREATE FOLDER("macintosh hd:") // ruta absoluta
CREATE FOLDER("lunes:martes") //ruta absoluta (un volumen debe llamarse lunes)
CREATE FOLDER(":lunes:martes") // ruta relativa
```

Extraer contenidos de la ruta de acceso

Puede manejar el contenido de la ruta utilizando los comandos **Path to object** y **Object to path**. En particular, al utilizar estos comandos, puede extraer de una ruta de acceso:

- un nombre de archivo,
- la ruta de la carpeta principal,
- la extensión del archivo o la carpeta.

Append document

Append document (doc {; tipo}) -> Resultado

Parámetro	Tipo		Descripción
doc	Cadena	➡	Nombre del documento o Ruta de acceso completa al documento o Cadena vacía para mostrar la caja de diálogo estándar de apertura de archivos
tipo	Cadena	➡	Lista de tipos de documentos a filtrar o "*" para no filtrar los documentos
Resultado	DocRef	↩	Número de referencia del documento

Descripción

El comando **Append document** hace lo mismo que **Open document**: permite abrir un documento en disco.

La única diferencia es que **Append document** define la ubicación del archivo al final del documento mientras que **Open document** lo hace al principio.

Para mayor información consulte la descripción del comando **Open document.**

Ejemplo

El siguiente ejemplo abre un documento existente llamado Nota, añade la cadena “y hasta pronto” seguida por un retorno de carro al final del documento, y cierra el documento. Si el documento ya contiene la cadena “Adiós”, el documento contendrá ahora la cadena “Adiós y hasta pronto”, seguido por un retorno de carro:

```
C_TIME(vhDocRef)
vhDocRef:=Append document("Nota.txt") ` Abrir el documento Nota
SEND PACKET(vhDocRef;" y hasta pronto"+Char(13)) ` Añadir la cadena
CLOSE DOCUMENT(vhDocRef) ` Cerrar el documento
```

CLOSE DOCUMENT

CLOSE DOCUMENT (docRef)

Parámetro	Tipo		Descripción
docRef	DocRef	➡	Número de referencia del documento

Descripción

CLOSE DOCUMENT cierra el documento especificado por *docRef*.

Cerrar un documento es la única forma de asegurar que los datos escritos en el archivo sean guardados. Debe cerrar todos los documentos abiertos por los comandos **Open document**, **Create document** o **Append document**.

Ejemplo

El siguiente ejemplo permite al usuario crear un nuevo documento, escribe la cadena “Hola” y cierra el documento:

```
C_TIME(vhDocRef)
vhDocRef:=Create document("")
If(OK=1)
   SEND PACKET(vhDocRef;"Hola") ` Escribe una palabra en el documento
   CLOSE DOCUMENT(vhDocRef) ` Cierra el documento
End if
```

Convert path POSIX to system

Convert path POSIX to system (rutaPosix {; *}) -> Resultado

Parámetro	Tipo		Descripción
rutaPosix	Texto	➡	Ruta de acceso POSIX
*	Operador	➡	Opción de codificación
Resultado	Texto	⮌	Ruta de acceso expresada en sintaxis sistema

Descripción

El comando **Convert path POSIX to system** convierte una ruta de acceso expresada con la sintaxis POSIX (Unix) en una ruta expresada con la sintaxis sistema.

Pase en el parámetro *rutaPosix* la ruta de acceso completa a un archivo o carpeta, expresada con la sintaxis POSIX. Esta ruta debe ser absoluta (debe comenzar con el carácter "/"). Debe pasar una ruta disco; no es posible pasar una ruta red (comenzando, por ejemplo con ftp://ftp.mysite.fr).

El comando devuelve la ruta de acceso completa del archivo o del archivo expresada en la sintaxis del sistema actual.

El parámetro opcional * permite indicar si el parámetro *rutaPosix* está codificado. Si este es el caso, debe pasar este parámetro, de lo contrario la conversión no será válida. El comando devuelve la ruta de acceso sin codificación.

Ejemplo 1

Ejemplos bajo Mac OS:

```
$path:=Convert path POSIX to system("/Volumes/machd/file 2.txt")
 //devuelve "machd:file 2.txt"
$path:=Convert path POSIX to system("/Volumes/machd/file%202.txt";*)
 //devuelve "machd:file 2.txt"
$path:=Convert path POSIX to system("/file 2.txt")
 //devuelve "machd:file 2.txt" si machd es el disco de inicio
```

Ejemplo 2

Ejemplos bajo Windows:

```
$path:=Convert path POSIX to system("c:/docs/file 2.txt")
 //devuelve "c:\\docs\\file 2.txt"
$path:=Convert path POSIX to system("c:/docs/file%202.txt";*)
 //devuelve "c:\\docs\\file 2.txt"
```

Convert path system to POSIX

Convert path system to POSIX (rutaSistema {; *}) -> Resultado

Parámetro	Tipo		Descripción
rutaSistema	Texto	➡	Ruta de acceso relativa o absoluta expresada en sintaxis sistema
*	Operador	➡	Opción de codificación
Resultado	Texto	↩	Absolute pathname expressed in POSIX syntax

Descripción

El comando **Convert path system to POSIX** convierte una ruta expresada con la sintaxis sistema en una ruta expresada con la sintaxis POSIX (Unix).

Pase en el parámetro *rutaSistema* la ruta de acceso a un archivo o carpeta, expresada con la sintaxis sistema (Mac OS o Windows). Esta ruta puede ser absoluta o relativa a la carpeta de la base (carpeta que contiene el archivo de estructura de la base). No es obligatorio que los elementos de la ruta existan realmente en el disco en el momento de la ejecución del comando (el comando no prueba la validez de la ruta de acceso).

El comando devuelve la ruta de acceso completa del archivo o de la carpeta expresada en la sintaxis POSIX. El comando siempre devuelve una ruta absoluta, sin importar el tipo de la ruta pasada en *rutaSistema*. Si pasa una ruta relativa en *rutaSistema*, 4D completa el valor devuelto al añadir la ruta de acceso a la carpeta de la base.

El parámetro opcional * permite definir la codificación de la ruta POSIX. Por defecto, **Convert path system to POSIX** no codifica los caracteres especiales de la ruta POSIX. Si pasa el parámetro *, los caracteres especiales se traducen (por ejemplo, "My folder" se convierte en "Mi%20carpeta").

Ejemplo 1

Ejemplos bajo Mac OS

```
$path:=Convert path system to POSIX("machd:file 2.txt")
 //machd es el disco de inicio
 //devuelve "/file 2.txt"
$path:=Convert path system to POSIX("disk2:file 2.txt")
 //disk2 es un disco adicional (sin inicio)
 //devuelve "/Volumes/disk2/file 2.txt"
$path:=Convert path system to POSIX("machd:file 2.txt";*)
 //devuelve "/file%202.txt"
$path:=Convert path system to POSIX(":resources:images")  //ruta relativa
 //devuelve "/User/mark/Documents/videodatabase/resources/images"
$path:=Convert path system to POSIX("resources:images")  //ruta relativa
 //devuelve "/resources/images"
```

Ejemplo 2

Ejemplo bajo Windows

```
$path:=Convert path system to POSIX("c:\docs\file 2.txt")
 `returns "c:/docs/file 2.txt"
$path:=Convert path system to POSIX("\\srv\tempo\file.txt")
 `returns "//srv/tempo/file.txt"
```

COPY DOCUMENT

COPY DOCUMENT (nomFuente ; nomDest {; nuevNombre} {; *})

Parámetro	Tipo		Descripción
nomFuente	Cadena	➡	Nombre del documento a copiar
nomDest	Cadena	➡	Nombre del documento copiado
nuevNombre	Cadena	➡	Nuevo nombre del archivo o carpeta copiado
*	Operador	➡	Reemplazar documento existente si lo hay

Descripción

El comando **COPY DOCUMENT** copia el documento especificado por *nomFuente* en la ubicación especificada por *nomDest*.

- **Copia de archivo**
  En este caso, el parámetro *nomFuente* debe contener una ruta de acceso completa del archivo, expresada con respecto a la raíz del volumen.
  El parámetro *nomDest* puede contener varios tipos de lugares:
  - una ruta de acceso completa del archivo expresada con respecto a la raíz del volumen: el archivo se copia en esta ubicación
  - un nombre de archivo o una ruta de acceso del archivo relativa: el archivo se copia en la carpeta de la base (las subcarpetas ya deben existir)
  - una ruta de acceso completa de la carpeta o una ruta relativa a la carpeta de la base (*nomDest* debe terminar con el separador de carpeta de la plataforma): el archivo se copia en la carpeta designada. Estas carpetas ya deben existir en el disco, no se crean.

  Se generará un error si ya hay un documento llamado *nomDest* a menos que haya especificado el parámetro opcional * que le indica a **COPY DOCUMENT** que borre y remplace el documento en la ubicación de destino en este caso.

- **Copia de carpeta**
  Para indicar que se designa una carpeta, las cadenas pasadas en *nomFuente* y *nomDest* deben terminar con un separador de carpetas de la plataforma. Por ejemplo, en Windows "C: \\Element\\" designa una carpeta y "C:\\Element" designa un archivo.
  Para copiar una carpeta, pase su ruta de acceso completa en *nomFuente*. Esta carpeta debe existir en el disco. Cuando una carpeta se define en el parámetro *nomFuente*, una carpeta también deberá designarse en el parámetro *nomDest*. Debe pasar una ruta de acceso completa de carpeta (cada elemento ya debe existir en el disco).
  Si ya existe una carpeta con el mismo nombre de la carpeta designada por el parámetro *nomFuente* en la ubicación definida por el parámetro *nomDest* y no está vacía, 4D verifica su contenido antes de copiar los elementos. Un error se genera cuando ya existe un archivo con el mismo nombre, a menos que haya pasado el parámetro opcional *, que indica al comando borrar y remplazar el archivo en la ubicación de destino.

Note que puede pasar un archivo en el parámetro *nomFuente* y una carpeta en el parámetro *nomDest*, para copiar un archivo en una carpeta.

El parámetro opcional *nuevoNombre*, si se pasa, permite renombrar el documento copiado a su ubicación de destino (archivo o carpeta). Cuando se pasa en el contexto de una copia de archivo, este parámetro remplaza el nombre (si lo hay) pasado en el parámetro *nomDest*.

Ejemplo 1

El siguiente ejemplo duplica un documento en su propia carpeta:

```
COPY DOCUMENT("C:\\CARPETA\\DocNombre";"C:\\CARPETA\\DocNombre2")
```

Ejemplo 2

El siguiente ejemplo copia un documento en la carpeta de la base (siempre y cuando **C:\\CARPETA** no sea la carpeta de la base):

```
COPY DOCUMENT("C:\\CARPETA \\DocNombre";"DocNombre")
```

Ejemplo 3

El siguiente ejemplo copia un documento de un volumen a otro:

```
COPY DOCUMENT("C:\\CARPETA \\DocNombre";"F:\\Archivos\\DocNombre.OLD")
```

Ejemplo 4

El siguiente ejemplo duplica un documento en su propia carpeta, sobrescribiendo una copia existente:

```
COPY DOCUMENT("C:\\CARPETA \\DocNombre";"C:\\CARPETA \\DocNombre2";*)
```

Ejemplo 5

Copia de un archivo en una carpeta específica conservando el mismo nombre:

```
COPY DOCUMENT("C:\\Projects\\DocName";"C:\\Projects\\")
```

Ejemplo 6

Copiar un archivo en una carpeta específica conservando el mismo nombrey remplazando el documento existente:

```
COPY DOCUMENT("C:\\Projects\\DocName";"C:\\Projects\\"; *)
```

Ejemplo 7

Copia de una carpeta en otra carpeta (ambas carpetas deben existir en el disco):

```
COPY DOCUMENT("C:\\Projects\\";"C\\Archives\\2011\\")
```

Ejemplo 8

Los siguientes ejemplos crean diferentes archivos y carpetas en la carpeta de la base (ejemplos bajo Windows). En cada caso, la carpeta "folder2" debe existir:

```
COPY DOCUMENT("folder1\\name1";"folder2\\")
 //crea el archivo "folder2/name1"

COPY DOCUMENT("folder1\\name1";"folder2\\" ; "new")
 //crea el archivo "folder2/new"

COPY DOCUMENT("folder1\\name1";"folder2\\name2")
 //crea el archivo "folder2/name2"

COPY DOCUMENT("folder1\\name1";"folder2\\name2";"new")
 //crea el archivo "folder2/new" (name2 se ignora)

COPY DOCUMENT("folder1\\" ; "folder2\\")
 //crea la carpeta "folder2/folder1/"

COPY DOCUMENT("folder1\\" ; "folder2\\" ; "new")
 //crea la carpeta "folder2/new/"
```

CREATE ALIAS

CREATE ALIAS (rutaObjetivo ; rutaAlias)

Parámetro	Tipo		Descripción
rutaObjetivo	Cadena	➡	Nombre o ruta de acceso al objetivo del alias/atajo
rutaAlias	Cadena	➡	Nombre o ruta de acceso completa del alias/del atajo a crear

Descripción

El comando **CREATE ALIAS** crea un alias (llamado "atajo" en Windows) del archivo o carpeta objetivo pasado en *rutaObjetivo*. El nombre y la ubicación son definidos por el parámetro *rutaAlias*.

Puede crear un alias de todo tipo de documento o de carpeta. El icono del alias será idéntico al del elemento objetivo. El icono contiene una pequeña flecha en la parte inferior izquierda. Bajo Mac OS, el nombre del icono se muestra en caracteres en itálica.

Este comando no asigna un nombre por defecto, el nombre tiene que pasarse en el parámetro *rutaAlias*. Si sólo pasa un nombre en este parámetro, el alias se crea en la carpeta activa actual (generalmente la carpeta que contiene el archivo de estructura).

Nota: bajo Windows, los atajos son archivos con extensión ".LNK" (invisible). Si esta extensión no se pasa, es añadida automáticamente por el comando.

Si se pasa una cadena vacía en *rutaObjetivo*, el comando no hace nada.

Ejemplo

Su base genera archivos de texto llamados "InformeNúmero" almacenados en la carpeta de la base. El usuario quiere crear atajos a estos informes y almacenarlos en una ubicación conveniente:

```
   `Método CREAR_INFORME
C_TEXT($vtInf)
C_STRING(250;$vtruta)
C_STRING(80;$vanombre)
C_TIME(vDoc)
C_INTEGER($NumInforme)

$vtInf:=...  `Crear informe
$NumInforme:=...  `Calculo del número del informe
$vanombre:="Informe"+String($NumInforme)+".txt"  `Nombre del archivo
vDoc:=Create document($vanombre)
If(OK=1)
   SEND PACKET(vDoc;$vtInf)
   CLOSE DOCUMENT(vDoc)
  `Añadir el alias
   CONFIRM("¿Crear un alias para este informe?")
   If(OK=1)
      $vtRuta:=Select folder("¿Dónde quiere crear el alias?")
      If(OK=1)
         CREATE ALIAS($vaNombre;$vtRuta+$vanombre)
         If(OK=1)
            SHOW ON DISK($vtRuta+$vanombre)
  `Mostrar la ubicación del alias
         End if
      End if
   End if
End if
```

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si el comando se ejecuta correctamente, si no toma el valor 0.

Create document

Create document (doc {; tipo}) -> Resultado

Parámetro	Tipo		Descripción
doc	Cadena	➡	Nombre del documento o Ruta de acceso completa del documento o Cadena vacía para mostrar caja de diálogo estándar de guardar archivos
tipo	Cadena	➡	Lista de los tipos de documentos a filtrar o "*" para no filtrar los documentos
Resultado	DocRef	⮌	Número de referencia del documento

Descripción

El comando **Create document** crea un nuevo documento y devuelve su número de referencia.

Pase el nombre o ruta completa del nuevo documento *documento*. Si *documento* ya existe en el disco, se sobrescribe. Sin embargo, si *documento* está bloqueado o abierto, se genera un error.

Si pasa una cadena vacía en *documento*, aparece una caja de diálogo estándar de registro de archivos y el usuario puede especifica el nombre del documento que quiere crear. Si cancela el diálogo, no se crea el documento; **Create document** devuelve una referencia de documento nula y la variable OK toma el valor 0.

Si el documento se crea correctamente y se abre, **Create document** devuelve su número de referencia y la variable OK toma el valor 1. El documento de la variable sistema Document se actualiza y devuelve la ruta de acceso completa del documento creado.

Create document crea por defecto un documento de tipo .TXT (Windows) o TEXT (Macintosh). Para crear otro tipo de documento, pase el parámetro *elTipo*.

En el parámetro *elTipo*, puede pasar uno o varios tipos de archivo con el fin de configurar la lista de tipos autorizados en la caja de diálogo. Puede pasar una lista de varios tipos separados por un ; (punto y coma). Para cada tipo definido, se añadirá una línea al menú de elección del tipo de caja de diálogo.

Bajo Mac OS, puede pasar un tipo Mac OS clásico (TEXT, APPL, etc.), o un tipo UTI (Uniform Tipo Identifier). Los tipos UTIs son definidos por Apple para cumplir con las necesidades de estandarización de tipos de archivos. Por ejemplo, "public.text" es el tipo UTI de los archivos de tipo texto. Para mayor información sobre UTIs, consulte la siguiente dirección: https://developer.apple.com/library/mac/#documentation/Miscellaneous/Reference/UTIRef/Articles/System-DeclaredUniformTypeIdentifiers.html (documentación en inglés).

Bajo Windows, puede pasar igualmente un tipo de archivo clásico Mac OS, 4D efectúa la correspondencia internamente, o la extensión de archivos (.txt, .exe, etc.). Note que bajo Windows, el usuario puede "forzar" la visualización de todos los tipos de archivos introduciendo *.* en la caja de diálogo. Sin embargo, en este caso, 4D efectuará una verificación suplementaria de los tipos de archivos seleccionados: si el usuario selecciona un tipo de archivo no autorizado, el comando devuelve un error.

Si no quiere restringir los archivos mostrados a uno o a más tipos, pase "*" (asterisco) o ".*" en *elTipo*.

En Windows pase una extensión de archivo Windows o un tipo de archivo Mac OS asociado con la ayuda del comando **_o_MAP FILE TYPES**. Si quiere crear un documento sin extensión, un documento con varias extensiones, o un documento con una extensión de más de tres caracteres, no utilice el parámetro *elTipo* y pase el nombre completo en *documento* (ver ejemplo 2).

Una vez haya creado y abierto un documento, puede escribir o leer los valores del documento utilizando los comandos **RECEIVE PACKET** y **SEND PACKET** que puede combinar con los comandos **Get document position** y **SET DOCUMENT POSITION** para acceder directamente a ciertas partes del documento.

No olvide llamar finalmente a **CLOSE DOCUMENT** para el documento.

Ejemplo 1

El siguiente ejemplo crea y abre un nuevo documento llamado Nota, escribe la cadena "Hola" y cierra el documento:

```
C_TIME(vhDoc)
vhDoc:=Create document("Nota.txt") ` Crear un nuevo documento llamado Nota
If(OK=1)
   SEND PACKET(vhDoc;"Hola") ` Escribir una palabra en el documento
   CLOSE DOCUMENT(vhDoc) ` Cerrar el documento
End if
```

Ejemplo 2

El siguiente ejemplo crea documentos con extensiones que no son estándar en Windows:

```
$vtMiDoc:=Create document("Doc.ext1.ext2") `Varias extensiones
$vtMiDoc:=Create document("Doc.shtml") `Extensión larga
$vtMiDoc:=Create document("Doc.") `Sin extensión (el punto "." es obligatorio)
```

Variables y conjuntos del sistema

Si el documento se crea correctamente, la variable sistema OK toma el valor 1 y la variable sistema Document contiene la ruta completa y el nombre del archivo *documento*.

CREATE FOLDER

CREATE FOLDER (rutaCarpeta {; Operador})

Parámetro	Tipo		Descripción
rutaCarpeta	Cadena	➡	Ruta de acceso a la nueva carpeta a crear
Operador	Operador	➡	Crear carpeta jerárquica

Descripción

El comando **CREATE FOLDER** crea una carpeta en función de la ruta de acceso que se pasa en *rutaCarpeta*.

Si pasa un nombre en *rutaCarpeta*, la carpeta se crea en la carpeta de la base.

En *rutaCarpeta*, también puede pasar una jerarquía de carpetas a partir de la raíz del volumen o de la carpeta de la base (en este caso, la cadena debe terminar con sun separador de carpeta).
Si omite el parámetro *, se genera un error y ninguna carpeta se crea si ninguna de las carpetas intermediarias existe.
Si pasa el parámetro *, **CREATE FOLDER** recrea la jerarquía de carpetas si es necesaria y no se genera ningún error. En este caso, puede pasar una ruta de acceso de documento en *rutaCarpeta*. Entonces se ignora el nombre del documento pero la jerarquía de carpetas especificada en *rutaCarpeta* se crea recursivamente.

Ejemplo 1

El siguiente ejemplo crea la carpeta “Archivos” en la carpeta de la base:

```
CREATE FOLDER("Archivos")
```

Ejemplo 2

El siguiente ejemplo crea la carpeta Archivos en la carpeta de la base, luego crea las subcarpetas “Enero” y “Febrero”:

```
CREATE FOLDER("Archivos")
CREATE FOLDER("Archivos\\Enero")
CREATE FOLDER("Archivos\\Febrero")
```

Ejemplo 3

El siguiente ejemplo crea la carpeta “Archivos” en la raíz del volumen C:

```
CREATE FOLDER("C:\\Archivos")
```

Ejemplo 4

Creación de la jerarquía de carpetas "C:\Archives\2011\Enero\":

```
CREATE FOLDER("C:\\Archives\\2011\\Enero\\";*)
```

Ejemplo 5

Creación de la subcarpeta "\Febrero\" en la carpeta existente "C:\Archives\":

```
CREATE FOLDER("C:\\Archives\\2011\\Febrero\\Doc.txt";*)
 // el archivo "Doc.txt" se ignora
```

DELETE DOCUMENT

DELETE DOCUMENT (doc)

Parámetro	Tipo		Descripción
doc	Cadena	➡	Nombre del documento o Ruta de acceso completa al documento

Descripción

El comando **DELETE DOCUMENT** borra el documento cuyo nombre se pasa en *documento*.

Si el nombre del documento o la ruta de acceso son incorrectos, se genera un error. Este también es el caso si trata de borrar un documento abierto.

DELETE DOCUMENT no acepta una cadena vacía en el parámetro *documento*. Si se utiliza una cadena vacía, la caja de diálogo de apertura de archivos no se muestra y se genera un error.

Advertencia: DELETE DOCUMENT puede borrar un archivo en disco. Esto incluye documentos creados con otras aplicaciones como también las aplicaciones. **DELETE DOCUMENT** debe usarse con extremo cuidado. La eliminación de un documento es una operación permanente y no puede deshacerse.

Ejemplo 1

El siguiente ejemplo borra el documento llamado Nota:

```
DELETE DOCUMENT("Nota") ` Borra el documento
```

Ejemplo 2

Ver el ejemplo del comando **APPEND DATA TO PASTEBOARD**.

Variables y conjuntos del sistema

La eliminación de un documento hace que la variable sistema tome el valor 1. Si **DELETE DOCUMENT** no puede borrar el documento, la variable sistema OK toma el valor 0.

DELETE FOLDER

DELETE FOLDER (carpeta {; opcionEliminacion})

Parámetro	Tipo		Descripción
carpeta	Cadena	➡	Nombre o ruta de acceso completa de la carpeta a borrar
opcionEliminacion	Entero largo	➡	Opción de eliminación de la carpeta

Descripción

El comando **DELETE FOLDER**borra la carpeta cuyo nombre o ruta completa se pasa en *carpeta*.
Por defecto, por razones de seguridad, si se omite el parámetro *opcionEliminacion*, **DELETE FOLDER** sólo permite carpetas vacías a eliminar. Si desea que el comando pueda eliminar carpetas no vacías, debe utilizar el parámetro *opcionEliminacion*. En *opcionEliminacion*, puede pasar una de las siguientes constantes, que se encuentra en el tema "**Documentos sistema**":

Constante	Tipo	Valor	Comentario
Delete only if empty	Entero largo	0	Elimina la carpeta sólo cuando está vacía
Delete with contents	Entero largo	1	Elimina la carpeta junto con todo su contenido

- Cuando se pasa Delete only if empty (0) o si se omite el parámetro *opcionEliminacion*:
  - La carpeta especificada en el parámetro *carpeta* solamente se borra si está vacía; de lo contrario, el comando no hace nada y se genera un error -47 (el archivo ya está abierto, o la carpeta no está vacía).
  - Si la carpeta especificada no existe, se genera el error -120 (Intento de acceso a un archivo con una ruta de acceso que especifica un directorio no existente).
- Cuando se pasa Delete with contents (1):
  - Se elimina la carpeta, junto con todo su contenido.
    Advertencia: incluso cuando esta carpeta y/o su contenido están bloqueados o en modo sólo lectura, si el usuario actual tiene los derechos de acceso adecuados, se eliminan.
  - Si esta carpeta, o cualquiera de los archivos que contiene, no se pueden eliminar, la eliminación se interrumpe tan pronto como se detecta el primer elemento inaccesible, y se devuelve un error (*). En este caso, la carpeta se puede eliminar sólo parcialmente. Cuando se cancela la eliminación, puede utilizar el comando [#cmd id="1015"/] para recuperar el nombre y la ruta del archivo que origina el error.
  - Si la carpeta especificada no existe, el comando no hace nada y ningún error se devuelve.

    (*) En Windows: -54 (Intento de abrir un archivo bloqueado para escritura)
    En OS X: -45 (El archivo está bloqueado o la ruta de acceso no es correcta)

Puede interceptar estos errores utilizando un método instalado por el comando **ON ERR CALL**.

DOCUMENT LIST

DOCUMENT LIST (nombreRuta ; documentos {; opciones})

Parámetro	Tipo		Descripción
nombreRuta	Cadena	➡	Ruta de acceso al volumen o a la carpeta
documentos	Array texto	⬅	Nombres de los documentos presentes en esta ubicación
opciones	Entero largo	➡	Opciones para crear la lista

Descripción

El comando **DOCUMENT LIST** llena el array de tipo Texto *documentos* con los nombres del los documentos ubicados en la ubicación pasada en *rutaAcceso*.

Nota: el parámetro *rutaAcceso* sólo acepta rutas de acceso absolutas.

Por defecto, si omite el parámetro *opciones*, sólo los nombres de los documentos se devuelven en el array *documentos*. Puede modificar este funcionamiento pasando en el parámetro *opciones*, una o más de las siguientes constantes, que se encuentran en el tema **Documentos sistema**:

Constante	Tipo	Valor	Comentario
Absolute path	Entero largo	2	El array documentos contiene las rutas de acceso absolutas
Ignore invisible	Entero largo	8	Los documentos invisibles no se lista
Posix path	Entero largo	4	El array *documentos* contiene las rutas de acceso al formato POSIX
Recursive parsing	Entero largo	1	El array documentos contiene los archivos y todas las subcarpetas de la carpeta especificada

Notas:

- Con la opción Recursive parsing en modo relativo (opción 1 únicamente), las rutas de los documentos ubicadas en las subcarpetas comienzan con los caracteres ":" o "\" dependiendo de la plataforma.
- Con la opción Posix path en modo relativo (opción 4 únicamente), las rutas no comienzan por "/".
- Con la opción Posix path en modo absoluto (opción 4 + 2), las rutas siempre comienzan por "/".

Si no hay documentos en la ubicación especificada, el comando devuelve un array vacío. Si la ruta de acceso pasada en *rutaAcceso* es invalida, **DOCUMENT LIST** genera un error de gestión de archivo que se puede interceptar utilizando un método **ON ERR CALL**.

Ejemplo 1

Lista de todos los documentos en una carpeta (sintaxis por defecto):

```
DOCUMENT LIST("C:\\";arrFiles)
```

-> arrFiles:
Text1.txt
Text2.txt

Ejemplo 2

Lista de todos los documentos en una carpeta en modo absoluto:

```
DOCUMENT LIST("C:\\";arrFiles; Absolute path)
```

-> arrFiles:
C:\Text1.txt
C:\Text2.txt

Ejemplo 3

Lista de todos los documentos en modo recursivo (relativo):

```
DOCUMENT LIST("C:\\";arrFiles;Recursive parsing)
```

-> arrFiles:
Text1.txt
Text2.txt
\Folder1\Text3.txt
\Folder1\Text4.txt

\Folder2\Text5.txt
\Folder2\Folder3\Picture1.png

Ejemplo 4

Lista de todos los documentos en modo recursivo absoluto:

```
DOCUMENT LIST("C:\\";arrFiles;Recursive parsing)
```

-> arrFiles:
C:\MyFolder\MyText1.txt
C:\MyFolder\MyText2.txt
C:\MyFolder\Folder1\MyText3.txt
C:\MyFolder\Folder1\MyText4.txt
C:\MyFolder\Folder2\MyText5.txt
C:\MyFolder\Folder2\Folder3\MyPicture1.png

Ejemplo 5

Lista de todos los documentos en modo recursivo Posix (relativo):
#code4D]DOCUMENT LIST("C:\\MyFolder\\";arrFiles;Recursive parsing+Posix path)[#/code4D]
-> arrFiles:
MyText1.txt
MyText2.txt
Folder1/MyText3.txt
Folder1/MyText4.txt
Folder2/MyText5.txt
Folder2/Folder3/MyPicture1.png

Document to text

Document to text (nomArchivo {; conjCaracteres {; modoRetorno}}) -> Resultado

Parámetro	Tipo		Descripción
nomArchivo	Cadena	➡	Nombre del documento o ruta al documento
conjCaracteres	Texto, Entero largo	➡	Nombre o número del conjunto de caracteres
modoRetorno	Entero largo	➡	Modo de procesamiento para las líneas de ruptura
Resultado	Texto	↩	Texto del documento

Descripción

El comando **Document to text** le permite recuperar el contenido de un archivo directamente en el disco en una variable texto 4D o campo texto.

En *nomArchivo*, pase el nombre o ruta de acceso del archivo a leer. El archivo debe existir en el disco, de lo contrario se genera un error. Puede pasar:

- sólo el nombre del archivo, por ejemplo "miArchivo.txt": en este caso, el archivo debe estar ubicado junto al archivo de estructura de la aplicación.
- una ruta de acceso relativa al archivo de estructura de la aplicación, por ejemplo, "\\docs\\miArchivo.txt" en Windows o ":docs: miArchivo.txt" en OS X.
- una ruta de acceso absoluta, por ejemplo, "c:\\app\\docs\\miArchivo.txt" en Windows o "MacHD:docs:miArchivo.txt" en OS X.

En *conjCaracteres*, se pasa el conjunto de caracteres a utilizar para la lectura de los contenidos. Puede pasar una cadena con el nombre estándar del conjunto (por ejemplo, "ISO-8859-1" o "UTF-8") o su ID MIBEnum (entero largo). Para más información sobre la lista de conjuntos de caracteres soportados por 4D, consulte la descripción del comando **CONVERT FROM TEXT**.

Si el documento contiene un Byte Order Mark (BOM), 4D utiliza el conjunto de caracteres que se ha definido en *conjCaracteres* (este parámetro entonces se ignora).

Si el documento no contiene un BOM y si se omite el parámetro *conjCaracteres,* 4D utiliza por defecto los siguientes conjuntos de caracteres:

- bajo Windows: ANSI
- bajo OS X: MacRoman

En breakMode, puede pasar un entero largo que indica el proceso a efectuar en los caracteres de fin de línea presentes en el documento. Puede pasar una de las siguientes constantes, del tema "**Documentos sistema**":

Constante	Tipo	Valor	Comentario
Document unchanged	Entero largo	0	Ningún proceso
Document with CR	Entero largo	3	Las líneas de ruptura se convierten al formato OS X: CR (*retorno de carro*)
Document with CRLF	Entero largo	2	Las líneas de ruptura se convierten al formato Windows: CRLF (*retorno de carro* + *salto de línea*)
Document with LF	Entero largo	4	Las líneas de ruptura se convierten al formato Unix: LF (*salto de línea*)
Document with native format	Entero largo	1	(Por defecto) las líneas de ruptura se convierten al formato nativo del sistema operativo: CR (retorno de carro) en OS X, CRLF (*retorno de carro* + *salto de línea*) en Windows

Por defecto, cuando se omite el parámetro *modoRetorno*, los saltos de línea se procesan en modo nativo (1).

Nota: este comando no modifica la variable OK. En caso de fallo, se genera un error que puede interceptar utilizando un método instalado por el comando **ON ERR CALL**.

Ejemplo

Dado el documento texto siguiente (los campos están separados por tabulaciones):

```
id    name    price    vat
3    4D Tags    99    19.6
```

Cuando ejecuta este código:

```
$Text:=Document to text("products.txt")
```

... obtiene:

```
// $Text = "id\tname\tprice\tvat\r\n3\t4D Tags\t99 \t19.6"
// \t = tab
// \r = CR
```

FOLDER LIST

FOLDER LIST (nombreRuta ; directorios)

Parámetro	Tipo		Descripción
nombreRuta	Cadena	➡	Ruta de acceso del volumen, directorio o carpeta
directorios	Array cadena	⬅	Nombres de los directorios presentes en esta ubicación

Descripción

El comando **FOLDER LIST** llena el array de tipo Texto o Alfa *directorios* con los nombres de las carpetas ubicadas en la ruta de acceso que se pasa en *rutaAcceso*.

Nota: debe pasar una ruta de acceso absoluta en el parámetro *rutaAcceso*.

Si no hay carpetas en la ubicación especificada, el comando devuelve un array vacío. Si la ruta de acceso que se pasa en *rutaAcceso* es incorrecta, **FOLDER LIST** genera un error del administrador de archivos que puede interceptar utilizando un método **ON ERR CALL**.

GET DOCUMENT ICON

GET DOCUMENT ICON (rutaDoc ; icono {; tamaño})

Parámetro	Tipo		Descripción
rutaDoc	Cadena	➡	Nombre o ruta de acceso del archivo del cual obtener el icono o cadena vacía para mostrar la caja de diálogo de apertura de archivos
icono	Campo imagen, Variable imagen	⬅	Icono de documentación
tamaño	Entero largo	⬅	Tamaño del icono (en píxeles)

Descripción

El comando **GET DOCUMENT ICON** devuelve en el campo o la variable imagen 4D *icono*, el icono del documento cuyo nombre o ruta de acceso se pasa en *rutaDoc*. *rutaDoc* puede especificar un archivo de todo tipo (ejecutable, documento, atajo o alias, etc.) o una carpeta.

Pase en *rutaDoc* la ruta de acceso absoluta del documento. Igualmente, puede pasar únicamente el nombre del documento o ruta de acceso relativa, en este caso el documento debe encontrarse en el directorio actual de la base (generalmente, la carpeta que contiene el archivo de estructura de la base).

Si pasa una cadena vacía en *rutaDoc*, aparece la caja de diálogo estándar de apertura de archivos, permitiendo al usuario seleccionar el archivo a leer. Una vez se valida la caja de diálogo, la variable sistema Document contiene la ruta de acceso completa del archivo seleccionado.

Pase en *icono* un campo o una variable imagen 4D. Después de la ejecución del comando, este parámetro contiene el icono del archivo (formato PICT).

El parámetro opcional *tamaño* permite indicar las dimensiones en píxeles del icono. Este valor representa el largo del cuadrado incluyendo el icono. Generalmente, los iconos se definen de 32x32 píxeles (“iconos largos”) o 16x16 píxeles (“iconos pequeños”). Si pasa 0 u omite este parámetro, el comando devuelve el icono más grande disponible.

Get document position

Get document position (docRef) -> Resultado

Parámetro	Tipo		Descripción
docRef	DocRef	➡	Número de referencia del documento
Resultado	Real	↩	Posición en el archivo (expresada en bytes) a partir del inicio del archivo

Descripción

Este comando sólo funciona en un documento abierto cuyo número de referencia se pasa en el parámetro *docRef*.

Get document position devuelve la posición, a partir del inicio del documento, donde ocurrirá la próxima lectura (**RECEIVE PACKET**) o escritura (**SEND PACKET**).

GET DOCUMENT PROPERTIES

GET DOCUMENT PROPERTIES (doc ; bloqueado ; invisible ; creado el ; creado a las ; modificado el ; modificado a las)

Parámetro	Tipo		Descripción
doc	Cadena	➡	Nombre del documento
bloqueado	Booleano	⬅	Bloqueado (True) o no bloqueado (False)
invisible	Booleano	⬅	Invisible (True) o visible (False)
creado el	Fecha	⬅	Fecha de creación
creado a las	Hora	⬅	Hora de creación
modificado el	Fecha	⬅	Fecha de la última modificación
modificado a las	Hora	⬅	Hora de la última modificación

Descripción

El comando **GET DOCUMENT PROPERTIES** devuelve información sobre el documento cuyo nombre o ruta se pasa en *documento*.

Después de la llamada:

- *bloqueado* devuelve True si el documento está bloqueado. Un documento bloqueado no puede modificarse.
- *invisible* devuelve True si el documento está oculto.
- *creado el* y *creado a las* devuelven la fecha y hora de creación del documento.
- *modificado el* y *modificado a las* devuelven la fecha y hora de la última modificación del documento.

Ejemplo

Usted ha creado una base de documentación y quiere exportar todos los registros creados en la base a un documento en disco. Como la base se actualiza regularmente, usted quiere escribir un algoritmo de exportación que cree o recree cada documento en el disco si el documento no existe o si el registro correspondiente ha sido modificado después de que el documento fue grabado por última vez. Por lo tanto, usted debe comparar la fecha y la hora de la modificación del documento (si la hay) con su registro correspondiente. Para ilustrar este ejemplo, utilizamos la siguiente tabla:

En lugar de guardar una fecha y una hora en cada registro, puede guardar un "marcador" que exprese el número de segundos transcurridos entre una fecha anterior arbitraria (en este ejemplo utilizamos el 1 de enero de 1995 a las 00:00:00) y la fecha y la hora en la que se guardó el registro.

En nuestro ejemplo, el campo *[Documentos]Marcador_creacion* contiene el marcador de creación del registro y el campo *[Documentos]Marcador_modificacion* contiene el marcador de la última modificación del registro.

El método de proyecto ***TimeStamp*** calcula el marcador de tiempo para una fecha y horas específicas o para la fecha y hora actual si no se pasan parámetros:

```
` Método de proyecto TimeStamp
` TimeStamp { ( date ; Time ) } -> Entero largo
` TimeStamp { ( date ; Time ) } -> Número de segundos desde el 1 de enero de 1995

C_DATE($1;$vdDate)
C_TIME($2;$vhTime)
C_LONGINT($0)

If(Count parameters=0)
   $vdDate:=Current date
   $vhTime:=Current time
Else
   $vdDate:=$1
   $vhTime:=$2
End if
$0:=(($vdDate-!01/01/95!)*86400)+$vhTime
```

Nota: utilizando este método, puede codificar todas las fechas y las horas desde *01/01/95* a las *00:00:00* a *01/19/2063* a las *03:14:07* lo que cubre el intervalo de enteros largos de *0* a *2^31* menos uno.

Por el contrario, los métodos de proyecto ***Time stamp to date*** y ***Time stamp to time*** permiten extraer la fecha y la hora almacenadas en un marcador:

```
` Método de proyecto Time stamp to date
` Time stamp to date ( Long ) -> Date
```

```
  ` Time stamp to date ( Time stamp ) -> Extracted date

C_DATE($0)
C_LONGINT($1)

$0:=!01/01/95!+($1\86400)

  ` Método de proyecto Time stamp to time
  ` Time stamp to time ( Entero largo) -> Fecha
  ` Time stamp to time ( Time stamp ) -> Fecha extraída

C_TIME($0)
C_LONGINT($1)

$0:=Time(Time string(†00:00:00†+($1%86400)))
```

Para asegurar que los marcadores de los registros se actualicen correctamente, sin importar la manera en que son creados o modificados, debemos aplicar esta regla utilizando el trigger de la tabla *[Documentos]*:

```
 // Trigger de la tabla [Documentos]
Case of
   :(Trigger event=Save New Record Event)
      [Documents]Creation Stamp:=Time stamp
      [Documents]Modification Stamp:=Time stamp
   :(Trigger event=Save Existing Record Event)
      [Documents]Modification Stamp:=Time stamp
End case
```

Una vez implementado en la base, tenemos todo lo que necesitamos para escribir el método de proyecto ***CREATE DOCUMENTATION***. Utilizamos **GET DOCUMENT PROPERTIES** y **SET DOCUMENT PROPERTIES** para administrar la fecha y hora de creación y modificación de los documentos.

```
  ` Método de proyecto CREATE DOCUMENTATION

C_STRING(255;$vsRuta;$vsDocRutaNombre;$vsDocNombre)
C_LONGINT($vlDoc)
C_BOOLEAN($vbOnWindows;$vbDoIt;$vbBloqueados;$vbInvisible)
C_TIME($vhDocRef;$vhCreadoalas;$vhModificadoalas)
C_DATE($vdCreadoen;$vdModificadoen)

If(Application type=4D Remote Mode)
  ` Si se está corriendo 4D Client, guarda los documentos
  ` localmente en el equipo Cliente donde se encuentra 4D Client
   $vsRuta:=Long name to path name(Application file)
Else
  ` De lo contrario, guardamos los documentos donde se ubica el archivo de datos
   $vsRuta:=Long name to path name(Data file)
End if
  ` Guardar los documentos en un directorio que llamamos arbitrariamente "Documentación"
$vsRuta:=$vsRuta+"Documentación"+Char(Directory symbol)
  ` Si este directorio no existe, lo crea
If(Test path name($vsRuta)#Is a folder)
   CREATE FOLDER($vsRuta)
End if
  ` Establecer la lista de documentos existentes
  ` porque tenemos que borrar los obsoletos, en otras palabras
  ` los documentos cuyos registros correspondientes han sido borrados.
ARRAY STRING(255;$asDocumento;0)
DOCUMENT LIST($vsRuta;$asDocumento)
  ` Selección de todos los registros de la tabla [Documentos]
ALL RECORDS([Documentos])
  ` Para cada registro
$vlNbRegistros:=Records in selection([Documentos])
$vlNbDocs:=0
$vbOnWindows:=On Windows
For($vlDoc;1;$vlNbRegistros)
  ` Suponemos que tendremos que recrear el documento en disco
   $vbDoIt:=True
  ` Calculo del nombre y ruta de acceso del documento
   $vsDocNombre:="DOC"+String([Documentos]Numero;"00000")
   $vsDocRutaNombre:=$vsRuta+$vsDocNombre
  ` ¿Ya existe este documento?
```

```
    If(Test path name($vsDocRutaNombre+".HTM")=Is a document)
  ` Si es así, eliminamos el documento de la lista de documentos
  ` que pueden ser eliminados
        $vlElem:=Find in array($asDocumento;$vsDocNombre+".HTM")
        If($vlElem>0)
            DELETE FROM ARRAY($asDocumento;$vlElem)
        End if
  ` ¿Se guardó el documento desúes de la última vez que se modificó el registro?
        GET DOCUMENT PROPERTIES($vsDocRutaNombre+".HTM";$vbBloq;$vbInvisible;$vdCreadoEn;$vhCreatedAt;
        $vdModificadoen;$vhModificadoalas)
        If(marcadorTiempos($vdModificadoen;$vhModificadoalas)>=[Documentos]Marcador_modificacion)
  ` Si es así, no necesitamos crear nuevamente el documento
            $vbDoIt:=False
        End if
    Else
  ` El documento no existe, colocar estas dos variables en cero de manera que
  ` sepamos que tenemos que computarlos antes de fijar las propiedades finales
  ` del documento
        $vdModificadoen:=!00/00/00!
        $vhModificadoalas:=†00:00:00†
    End if
  `¿Necesitamos crear nuevamente el documento?
    If($vbDoIt)
  ` Si es así, incrementar el número de documentos actualizados
        $vlNbDocs:=$vlNbDocs+1
  ` Eliminar el documento si ya existe
        DELETE DOCUMENT($vsDocRutaNombre+".HTM")
  ` Y lo crea nuevamente
        If($vbOnWindows)
            $vhDocRef:=Create document($vsDocRutaNombre;"HTM")
        Else
            $vhDocRef:=Create document($vsDocRutaNombre+".HTM")
        End if
        If(OK=1)
  ` Escriba acá los contenidos del documento
            CLOSE DOCUMENT($vhDocRef)
            If($vdModificadoen=!00/00/00!)
  ` El documento no existía, definir los valores correctos para la fecha y hora de modificación
                $vdModificadoen:=Current date
                $vhModificadoalas:=Current time
            End if
  ` Cambiar las propiedades del documento de manera que su fecha y hora de creación
  ` sean iguales a la del registro correspondiente
            SET DOCUMENT PROPERTIES($vsDocRutaNombre+".HTM";$vbBloq;$vbInvisible;
            Marcador fecha([Documentos]Marcador_creacion);
            Marcador hora([Documentos]Marcador_creacion);
            $vdModificadoen;$vhModificadoalas)
        End if
    End if
  ` Sólo para saber que está pasando
    SET WINDOW TITLE("Proceso del documento "+String($vlDoc)+" of "+String($vlNbRegistros))
    NEXT RECORD([Documentos])
End for
  ` Eliminación de documentos obsoletos, en otras palabras
  ` aquellos que están en el array $asDocumento
For($vlDoc;1;Size of array($asDocumento))
    DELETE DOCUMENT($vsRuta+$asDocumento{$vlDoc})
    SET WINDOW TITLE("Eliminación de documento obsoleto: "+Char(34)+$asDocumento{$vlDoc}+Char(34))
End for
  ` Listo
ALERT("Número de documentos procesados: "+String($vlNbRegistros)+Char(13)+"Número de documentos actualizados: "
+String($vlNbDocs)+Char(13)+"Número de documentos borrados: "+String(Size of array($asDocumento)))
```

Get document size

Get document size (document {; *}) -> Resultado

Parámetro	Tipo		Descripción
document	Cadena, DocRef	➡	Número de referencia del documento o Nombre del documento
*	Operador	➡	En Mac OS únicamente: -si se omite, tamaño del data fork - si se especifica, tamaño del resource fork
Resultado	Real	↩	Tamaño (expresado en bytes) del documento

Descripción

El comando **Get document size** devuelve el tamaño de un documento, expresado en bytes.

Si el documento está abierto, pase su número de referencia en *documento*.
Si el documento no está abierto, pase su nombre o ruta en *documento*.

En Macintosh, si no pasa el parámetro opcional *, se devuelve el tamaño del data fork. Si no pasa el parámetro *, se devuelve el tamaño del resource fork.

Get localized document path

Get localized document path (rutaRelativa) -> Resultado

Parámetro	Tipo		Descripción
rutaRelativa	Texto	➡	Ruta de acceso relativa del documento del cual obtener la versión localizada
Resultado	Texto	⮌	Ruta de acceso absoluta del documento localizado

Descripción

El comando **Get localized document path** devuelve la ruta completa (absoluta) de un documento designado por *rutaRelativa* y ubicado en la carpeta xxx.lproj.

Este comando debe usarse dentro de una arquitectura de aplicación multi-lenguaje basada en la presencia de una carpeta **Resources** y de las subcarpetas xxx.lproj (xxx representa un idioma). Con esta arquitectura, 4D soporta automáticamente archivos localizados de tipo XLIFF así como las imágenes, pero es posible que deba usar el mismo mecanismo para otros tipos de archivos.

Pase en *rutaRelativa* la ruta de acceso relativa del documento a buscar. La ruta especificada debe ser relativa al primer nivel de la carpeta "xxx.lproj" de la base de datos. El comando devolverá una ruta de acceso completa utilizando la carpeta "xxx.lproj" correspondiente al idioma actual de la base de datos.

Nota: el idioma actual es definido automáticamente por 4D en función del contenido de la carpeta Resources (vea el comando **Get database localization**), o vía el comando **SET DATABASE LOCALIZATION**.

Puede expresar el contenido del parámetro rutaRelativa utilizando una sintaxis POSIX o sistema. Por ejemplo:

- xsl/log.xsl (sintaxis POSIX: utilizable bajo Mac OS o Windows)
- xsllog.xsl (Windows únicamente)
- xsl:log.xsl (Mac OS únicamente)

La ruta de acceso absoluta devuelta por el comando siempre se expresa en la sintaxis del sistema.

4D Server: en modo remoto, el comando devuelve la ruta de la carpeta Resources en la máquina cliente si el comando se llama desde un proceso cliente.

4D busca el archivo respetando una secuencia que permite tratar todos los casos de aplicaciones en varios idiomas. En cada paso, 4D comprueba la presencia de rutaRelativa en la carpeta correspondiente al idioma y devuelve la ruta completa cuando tiene éxito. Si *rutaRelativa* no se encuentra o si la carpeta no existe, 4D pasa al siguiente paso. Aquí están las carpetas para cada una de las diferentes etapas de búsqueda:

Lenguaje actual (por ejemplo: fr-ca)
Lenguaje actual sin región (por ejemplo: fr)
Lenguaje cargado por defecto al inicio (por ejemplo: en-ga)
Lenguaje cargado por defecto al inicio sin región (es por ejemplo: es)
Primera carpeta .lproj encontrada (por ejemplo: en.lproj)
Primer nivel de la carpeta Resources

Si *rutaRelativa* no se encuentra en ninguna de estas ubicaciones, el comando devuelve una cadena vacía.

Ejemplo

Para transformar un archivo XML en HTML, queremos utilizar un archivo de transformación "log.xsl". Este archivo difiere dependiendo del idioma actual. Usted quiere conocer la ruta del archivo "log.xsl" a utilizar.
Estos son los contenidos de la carpeta Resources:

Para utilizar un archivo .xsl adaptado al lenguaje actual, sólo debe pasar:

```
$myxsl:=Get localized document path("xsl/log.xsl")
```

Si el lenguaje actual es, por ejemplo, francés canadiense (fr-ca), el comando devuelve:

- bajo Windows: C:users…… resources_ca.lprojxsllog.xsl
- bajo Mac OS: "HardDisk:users:…:…:…:resources:fr_ca.lproj:xsl:log.xsl"

MOVE DOCUMENT

MOVE DOCUMENT (rutaFuente ; rutaDest)

Parámetro	Tipo		Descripción
rutaFuente	Cadena	➡	Ruta de acceso completa al documento existente
rutaDest	Cadena	➡	Ruta de acceso de destino

Descripción

El comando **MOVE DOCUMENT** mueve o renombra un documento.

Pase la ruta de acceso completa al documento existente en *rutaFuente* y el nuevo nombre y/o ubicación del documento en *rutaDest*.

Advertencia: utilizando **MOVE DOCUMENT**, puede mover un documento desde y hacia cualquier directorio en el mismo volumen. Si quiere mover un documento entre dos volúmenes diferentes, utilice el comando **COPY DOCUMENT** para "mover" el documento luego borre la copia original con el comando **DELETE DOCUMENT**.

Ejemplo 1

El siguiente ejemplo renombra el documento **DocNombre**:

```
MOVE DOCUMENT("C:\\CARPETA\\DocNombre";"C:\\CARPETA\\NewDocNombre")
```

Ejemplo 2

El siguiente ejemplo mueve y renombre el documento **DocNombre**:

```
MOVE DOCUMENT("C:\\CARPETA1\\DocNombre";"C:\\CARPETA2\\NewDocNombre")
```

Ejemplo 3

El siguiente ejemplo mueve el documento **DocName**:

```
MOVE DOCUMENT("C:\\CARPETA1\\DocNombre";"C:\\CARPETA2\\DocNombre")
```

Nota: en los dos últimos ejemplos, debe existir la carpeta de destino *"C:\\CARPETA2"*. El comando **MOVE DOCUMENT** sólo mueve un documento; no crea carpetas.

Object to path

Object to path (objRuta) -> Resultado

Parámetro	Tipo		Descripción
objRuta	Objeto	➡	Objeto que describe los contenidos de una ruta
Resultado	Texto	⮌	Nombre de ruta

Descripción

El comando **Object to path** devuelve un nombre de ruta (cadena) en función de la información de ruta que pasó en el parámetro *objRuta*. Las siguientes rutas son soportadas:

- Ruta del sistema (Windows o macOS) o ruta Posix. El tipo de ruta está definido por el último carácter de la propiedad parentFolder (ver a continuación).
- Ruta relativa o ruta absoluta (ver **Ruta de acceso absoluta o relativa** para más información).

En *objRuta*, pase un objeto que define la ruta que desea generar. Debe contener las siguientes propiedades:

Propiedad	Tipo	Descripción
parentFolder	Texto	Información del directorio para la ruta. El último carácter debe ser un separador de carpeta. El comando usa este carácter para conocer el tipo de ruta. Si se trata de un separador Posix ("/"), la ruta se crea con los separadores Posix; de lo contrario, se usa el separador del sistema.
name	Texto	Nombre final de archivo o carpeta de la ruta especificada sin extensión.
extension	Texto	Extensión del nombre final de archivo o carpeta. Comienza con "." (puede ser omitido). Cadena vacía "" si no hay extensión.
isFolder	Booleano	True si el nombre es un nombre de carpeta, de lo contrario, false (el valor predeterminado es false)

Por lo general, *objRuta* se generará con el comando **Path to object** , sin embargo, el objeto se puede generar con cualquier medio. Tenga en cuenta que **Object to path** solo maneja cadenas. Tampoco comprueba si la ruta es válida con respecto al tipo de ruta, ni la existencia real de ningún archivo o carpeta.

Ejemplo

Queremos duplicar y cambiar el nombre de un archivo en su propia carpeta

```
C_OBJECT($o)
$o:=New object
C_TEXT($path)
$path:="C:\\MyDocs\\file.txt"

$o:=Path to object($path)
$o.name:=$o.name+"_copy"
COPY DOCUMENT($path;Object to path($o))
```

Open document

Open document (doc {; tipo}{; modo}) -> Resultado

Parámetro	Tipo		Descripción
doc	Cadena	➡	Nombre del documento o Ruta de acceso completa al documento o Cadena vacía para mostrar la caja de diálogo
tipo	Cadena	➡	Lista de los tipos de documentos a filtrar o "*" para no filtrar los documentos
modo	Entero largo	➡	Modo de apertura del documento
Resultado	DocRef	↩	Número de referencia del documento

Descripción

El comando **Open document** abre el documento cuyo nombre o ruta de acceso se pasa en *documento*.

Si pasa una cadena vacía en *document*, aparece una caja de diálogo estándar de apertura de archivos y el usuario puede seleccionar el documento a abrir. Si cancela el diálogo, no se abre el documento; **Open document** devuelve una referencia de documento nula y la variable OK toma el valor 0.

- Si el documento se abre correctamente, **Open document** devuelve su número de referencia y la variable OK toma el valor 1.
- Si el documento está abierto en lectura y se omite el parámetro *modo*, **Open document** abre el documento en modo lectura/escritura y la variable OK toma el valor 1.
- Si el documento ya está abierto en modo escritura/lectura y usted intenta abrirlo en modo Escritura, se genera el error (-43). Sin embargo puede abrirlo en modo lectura únicamente, luego la variable OK toma el valor 1.
- Si el documento no existe, se genera un error.

En el parámetro *elTipo*, pase el o los tipo(s) de archivo(s) que pueden ser seleccionados en la caja de diálogo de apertura. Puede pasar una lista de varios tipos separados por un ; (punto y coma). Para cada tipo definido, se añadirá una línea en el menú de elección del tipo de caja de diálogo.

Bajo Mac OS, puede pasar el tipo Mac OS clásico (TEXT, APPL, etc.), o un tipo UTI (Uniform Tipo Identifier). Los tipos UTIs son definidos por Apple para responder a las necesidades de estandarización de los tipos de archivos. Por ejemplo, "public.text" es el tipo UTI de los archivos de tipo texto. Para mayor información sobre los UTIs, consulte Introduction to Uniform Type Identifiers Overview page en el sitio web *developer.apple.com* (documentación en inglès).

Bajo Windows, también puede pasar un tipo de archivo clásico Mac OS, 4D efectúa la correspondencia interna, o la extensión de los archivos (.txt, .exe, etc.). Note que bajo Windows, el usuario puede "forzar" la visualización de todos los tipos de archivos introduciendo *.* en la caja de diálogo. Sin embargo, en este caso, 4D llevará a cabo una verificación adicional de los tipos de archivos seleccionados: si el usuario selecciona un tipo de archivo no autorizado, el comando devuelve un error.
Si no quiere restringir los archivos mostrados a uno o varios tipos, pase la cadena "*" (asterisco) o ".*" en *elTipo*.

El parámetro opcional *modo* permite definir el modo de apertura del archivo *documento*. Están disponibles cuatro modos de apertura. 4D ofrece las siguientes constantes predefinidas, ubicadas en el tema **Documentos sistema**:

Constante	Tipo	Valor
Read and Write	Entero largo	0
Write Mode	Entero largo	1
Read Mode	Entero largo	2
Get Pathname	Entero largo	3

Si un documento está abierto, **Open document** se ubica inicialmente al comienzo del documento, mientras **Append document** se ubica al final del documento.

Una vez haya abierto un documento, puede leer y escribir en el documento utilizando los comandos **RECEIVE PACKET** y **SEND PACKET** que puede combinar con los comandos **Get document position** y **SET DOCUMENT POSITION** para acceder directamente a cualquier parte del documento.

No olvide llamar finalmente a **CLOSE DOCUMENT** para el documento.

Ejemplo 1

El siguiente ejemplo abre un documento existente llamado Note, escribe la cadena "Good-bye" en el documento y lo cierra. Todo contenido existente en el documento se sobrescribirá:

```
C_TIME(vhDoc)
vhDoc:=Open document("Note.txt";Read and Write)  //Abrir el documento Note
If(OK=1)
   SEND PACKET(vhDoc;"Good-bye")  //Escribe una palabra en el documento
   CLOSE DOCUMENT(vhDoc)  //Cierra el documento
End if
```

Ejemplo 2

Puede leer un documento incluso si está abierto en modo escritura:

```
vDoc:=Open document("PassFile";"TEXT")  ` El archivo está abierto
  ` Antes de cerrar el archivo, es posible consultarlo en modo de solo lectura:
```

```
vRef:=Open document("PassFile";"TEXT";Read Mode)
```

Variables y conjuntos del sistema

Si el documento se abre correctamente, la variable sistema OK toma el valor 1; de lo contrario, toma el valor 0. Después de la llamada, la variable sistema Document contiene el nombre completo del documento.

Si pasa el valor 3 en *modo*, la función devuelve ?00:00:00? (sin referencia de documento). El documento no se abre pero las variables sistema Document y OK se actualizan:

- OK toma el valor 1.
- Document contiene la ruta de acceso y el nombre del archivo *document*.

Nota: si no se encuentra el archivo definido en *documento* o si pasa una cadena vacía en *documento*, aparece una caja de diálogo de apertura de archivos. Si el usuario elije un documento y hace clic en el botón OK, *documento* define la ruta al documento seleccionado por el usuario y OK toma el valor 1. Si el usuario hace clic en el botón Cancelar, OK toma el valor 0.

Path to object

Path to object (Ruta {; tipoRuta}) -> Resultado

Parámetro	Tipo		Descripción
Ruta	Texto	➡	Nombre de la ruta
tipoRuta	Entero largo	➡	Tipo de sintaxis de ruta: Sistema (por defecto) o Posix
Resultado	Objeto	⬅	Objeto que describe los contenidos de la ruta

Descripción

El comando **Path to object** devuelve un objeto que contiene las propiedades específicas de la *ruta* que pasó en el parámetro.
De forma predeterminada, si omite el parámetro *tipoRuta*, se asumirá que pasó una *ruta* del sistema que contiene separadores del sistema ("\" en Windows, ":" en macOS). Si pasó una *ruta* Posix que contiene separadores de Posix ("/") o desea expresar el tipo de ruta, pase una de las siguientes constantes en el parámetro *tipoRuta*:

Constante	Tipo	Valor	Comentario
Path is POSIX	Entero largo	1	La ruta se expresa utilizando la sintaxis de Posix
Path is system	Entero largo	0	(Por defecto) La ruta se expresa utilizando la sintaxis actual del sistema (Windows o macOS)

El comando devuelve un objeto resultante de analizar la *ruta*. Las siguientes propiedades están disponibles:

Propiedad	Tipo	Descripción
parentFolder	Texto	Información del directorio para la ruta. El último caracter es siempre un separador de carpetas.
name	Texto	Nombre del archivo o carpeta final de la ruta especificada, sin extensión.
extension	Texto	Extensión del nombre final de archivo o carpeta. Siempre comienza por ".". Cadena vacía "" si no hay extensión.
isFolder	Booleano	True si el nombre es un nombre de carpeta, de lo contrario, de lo contrario (el valor predeterminado es false)

Se supondrá que pasó una ruta de carpeta si el último carácter de la *ruta* es un separador correspondiente al tipo de ruta (por ejemplo, "\" en Windows). De lo contrario, se supondrá que pasó un nombre de archivo.
La extensión, si no está vacía, se devuelve independientemente de si la ruta representa un archivo o una carpeta. En este caso, debe concatenar el nombre y la extensión para recuperar el nombre completo.

Tenga en cuenta que **Path to object** solo maneja cadenas. Tampoco comprueba si la ruta es válida con respecto al tipo de ruta, ni a la existencia real de ningún archivo o carpeta.

Ejemplo 1

Los siguientes ejemplos muestran varios resultados con rutas de archivos:

```
C_OBJECT($o)
$o:=Path to object("C:\\first\\second\\fileZ") //en Windows
 //$o.parentFolder="C:\\first\\second\\"
 //$o.name="fileZ"
 //$o.extension=""
 //$o.isFolder=false
```

```
C_OBJECT($o)
$o:=Path to object("osx:Users:john:Documents:Comments.text) //en macOS
 //$o.parentFolder="osx:Users:john:Documents:"
 //$o.name="Comments"
 //$o.extension=".text"
 //$o.isFolder=false
```

```
C_OBJECT($o)
$o:=Path to object("\\images\\jan\\pict1.png";Path is system) //en Windows
 //$o.parentFolder="\\images\\jan\\"
 //$o.name="pict1"
 //$o.extension=".png"
 //$o.isFolder=false
```

Definiendo una ruta a una carpeta:

```
C_OBJECT($o)
$o:=Path to object("osx:Users:oscargoldman:Desktop:Databases:") //macOS
 //$o.parentFolder="osx:Users:oscargoldman:Desktop:"
```

```
 //$o.name="Databases"
 //$o.extension=""
 //$o.isFolder=True
```

```
C_OBJECT($o)
$o:=Path to object("C:\\4D\\Main\\216410\\64\\4D\\4D.user\\")  //windows
 //$o.parentFolder="C:\\4D\\Main\\216410\\64\\4D\\"
 //$o.name="4D"
 //$o.extension=".user"
 //$o.isFolder=true
```

```
C_OBJECT($o)
$o:=Path to object("/first/second.bundle/";Path is POSIX)
 //$o.parentFolder="/first/"
 //$o.name="second"
 //$o.extension=".bundle"
 //$o.isFolder=true
```

Si la ruta es un directorio raíz, *parentFolder* está vacío:

```
C_OBJECT($o)
$o:=Path to object("C:\\")  //en windows
 //$o.parentFolder=""
 //$o.name="c:"
 //$o.extension=""
 //$o.isFolder=true
```

```
C_OBJECT($o)
$o:=Path to object("osx:")  //en macOS
 //$o.parentFolder=""
 //$o.name="osx"
 //$o.extension=""
 //$o.isFolder=true
```

Si la última parte de la ruta es ".something", se considera como un nombre de archivo:

```
C_OBJECT($o)
$o:=Path to object("/folder/.invisible";Path is POSIX)
 //$o.parentFolder="/folder/"
 //$o.name=".invisible"
 //$o.extension=""
 //$o.isFolder=false
```

Ejemplo 2

Puede combinar este comando con **Object to path** para cambiar el nombre de un archivo en una ruta:

```
C_OBJECT($o)
C_TEXT($path)
$o:=Path to object("C:\\4D\\resources\\images\\4D.jpg")
 //$o.parentFolder="C:\\4D\\resources\\images\\"
 //$o.name="4D"
 //$o.extension=".jpg"
 //$o.isFolder=false

$o.name:="4DOld"
$path:=Object to path($o)
 //$path="C:\4D\resources\images\4DOld.jpg"
```

Ejemplo 3

Desea saber la cantidad de subcarpetas en una ruta:

```
C_OBJECT($o)
C_TEXT($path)
```

```
C_LONGINT($vCount)
$path:=Select folder  //let the user select a folder
$o:=Path to object($path)
Repeat
    $o:=Path to object($o.parentFolder)
    $vCount:=$vCount+1
Until($o.parentFolder="")
ALERT("The path depth is: "+String($count))
```

RESOLVE ALIAS

RESOLVE ALIAS (rutaAlias ; rutaObjetivo)

Parámetro	Tipo		Descripción
rutaAlias	Cadena	⇒	Nombre o ruta de acceso completa del alias/atajo
rutaObjetivo	Cadena	⇐	Nombre o ruta de acceso completa del objetivo del alias/atajo

Descripción

El comando **RESOLVE ALIAS** devuelve la ruta completa del archivo o carpeta objetivo de un alias (llamado atajo bajo Windows).

En *rutaAlias* se pasa el nombre o ruta de acceso completa del alias.

Una vez ejecutado el comando, la variable *rutaObjetivo* contiene la ruta de acceso completa al archivo o carpeta del alias y la variable sistema OK toma el valor 1.

Si la ruta se pasa en *rutaAlias* corresponde a un archivo y no a un alias, *rutaObjetivo* devuelve la ruta de acceso del archivo y la variable sistema OK toma el valor 0.

Variables y conjuntos del sistema

Si *rutaAlias* especifica un alias/atajo, la variable sistema OK toma el valor 1. Si *rutaAlias* especifica un archivo estándar, la variable sistema OK toma el valor 0.

Select document

Select document (directorio ; tiposArchivos ; titulo ; opciones {; seleccionados}) -> Resultado

Parámetro	Tipo		Descripción
directorio	Texto, Entero largo	➡	• Ruta de acceso del directorio para mostrar por defecto en la caja de diálogo de selección o • Cadena vacía para mostrar la carpeta del usuario por defecto (“Mis documentos” bajo Windows, “Documentos” bajo Mac OS), o • Número de la ruta de acceso memorizada
tiposArchivos	Texto	➡	Lista de los tipos de documentos a filtrar, o "*" para no filtrar los documentos
titulo	Texto	➡	Título de la caja de diálogo de selección
opciones	Entero largo	➡	Opciones de selección
seleccionados	Array texto	⬅	Array que contiene la lista de rutas de acceso + los nombres de los archivos seleccionados
Resultado	Cadena	↩	Nombre del archivo seleccionado (primer archivo de la lista en caso de selección múltiple)

Descripción

El comando **Select document** muestra una caja de diálogo estándar de apertura de documentos, permitiendo al usuario definir uno o más archivos y devolver el nombre y/o ruta de acceso completa del o de los archivo(s) seleccionado(s).

El parámetro *directorio* indica la carpeta cuyo contenido debe mostrarse inicialmente en la caja de diálogo de apertura de documentos. Puede pasar tres tipos de valores:

- un texto que contiene la ruta de acceso completa del directorio a mostrar.
- una cadena vacía ("") para mostrar la carpeta del usuario por defecto del sistema (“Mis documentos” bajo Windows, “Documentos” bajo Mac OS).
- un número de ruta de acceso memorizada (de 1 a 32 000) para mostrar la carpeta asociada.
  Con este principio, puede almacenar en memoria la ruta de acceso de la carpeta abierta en el momento en que el usuario hace clic en el botón de selección, en otras palabras, la carpeta seleccionada por el usuario. Durante la primera llamada de un número arbitrario (por ejemplo, 5) el comando muestra la carpeta usuario por defecto del sistema operativo (equivalente a pasar una cadena vacía). El usuario podrá navegar las carpetas en el disco duro. Cuando el usuario hace clic en el botón de selección, la ruta de acceso se memoriza y asocia al número 5. Durante las siguientes llamadas al número 5, la ruta de acceso memorizada será utilizada por defecto. En caso de que se seleccione una nueva ubicación, la ruta número 5 se actualiza.
  Este mecanismo le permite memorizar hasta 32 000 rutas de acceso. Bajo Windows, cada ruta se conserva durante la sesión únicamente. Bajo Mac OS, las rutas son conservadas por el sistema y permanecen almacenadas de una sesión a otra.

Nota: este mecanismo es el mismo al utilizado por el comando **Select folder**. Los números de las rutas de acceso memorizadas son compartidos por ambos comandos.

Pase en el parámetro *tiposArchivos* el o los tipo(s) de archivo(s) que pueden ser seleccionados en la caja de diálogo de apertura. Puede pasar una lista de varios tipos separados por un ; (punto y coma). Para cada tipo definido, una fila será añadida en el menú de selección del tipo de la caja de diálogo.

- Bajo Mac OS, puede pasar un tipo Mac OS clásico (TEXT, APPL, etc.), o un tipo UTI (Uniform Tipo Identifier). Los tipos UTIs son definidos por Apple para cumplir con las necesidades de estandarización de tipos de archivos. Por ejemplo, "public.text" es el tipo UTI de los archivos de tipo texto. Para mayor información sobre UTIs, consulte Uniform Type Identifier Concepts page en el sitio web *developer.apple.com* (documentación en ingles).

- Bajo Windows, puede pasar igualmente un tipo de archivo clásico Mac OS, 4D efectúa la correspondencia internamente o la extensión de archivos (.txt, .exe, etc.). Note que bajo Windows, el usuario puede “forzar” la visualización de todos los tipos de archivos introduciendo *.* en la caja de diálogo. Sin embargo, en este caso, 4D efectuará una verificación adicional de los tipos de archivos seleccionados: si el usuario selecciona un tipo de archivo no autorizado, el comando devuelve un error.

Si no quiere restringir los archivos mostrados a uno o a más tipos, pase "*" (asterisco) o ".*" en *tiposArchivos*.

Pase en el parámetro *titulo* la etiqueta que debe aparecer en la caja de diálogo. Por defecto, si pasa una cadena vacía, se muestra la etiqueta “Abrir”.

El parámetro *opciones* permite especificar funciones avanzadas autorizadas en la caja de diálogo de apertura. 4D ofrece las siguientes constantes predefinidas en el tema **Documentos sistema**. Puede pasar una constante o una combinación de constantes.

Constante	Tipo	Valor	Comentario
Alias selection	Entero largo	8	Autoriza la selección de atajos (Windows) o de alias (Mac OS) como documentos. Por defecto, si no se utiliza esta constante no se utiliza, cuando se seleccione un alias o atajo, el comando devolverá la ruta de acceso del elemento objetivo. Cuando pase la constante, el comando devuelve la ruta del alias o del atajo.
File name entry	Entero largo	32	Autoriza al usuario a introducir un nombre de archivo en una caja de diálogo "Guardar como". No se guardan los archivos, el desarrollador debe crear un archivo en respuesta a esta acción (la variable sistema Documento se actualiza). En este contexto, el parámetro directorio puede contener la ruta a un archivo en lugar de a un directorio. El nombre del archivo se utilizará como nombre de archivo sugerido en el campo de texto Guardar como. El directorio padre se utilizará como ruta por defecto.
Multiple files	Entero largo	1	Autoriza la selección simultánea de varios archivos utilizando las combinaciones **Mayús+clic** (selección adyacente) y **Ctrl+clic** (Windows) o **Comando+clic** (Mac OS). En este caso, el parámetro *seleccionado*, si se pasa, contiene la lista de todos los archivos seleccionados. Por defecto, si esta constante no se utiliza, el comando no permitirá la selección de archivos múltiples.
Package open	Entero largo	2	(Mac OS únicamente): autoriza la apertura de paquetes como carpetas y por lo tanto la visualización/selección de sus contenidos. Por defecto, si no se utiliza esta constante, el comando no permitirá la apertura de paquetes.
Package selection	Entero largo	4	(Mac OS únicamente): autoriza la selección de paquetes como entidades. Por defecto, si no se utiliza esta constante, el comando no permitirá la selección de paquetes como tales.
Use sheet window	Entero largo	16	(Mac OS únicamente): muestra la caja de diálogo de selección en forma de una ventana hoja (esta opción se ignora en Windows). Las ventanas hojas son específicas para la interfaz Mac OS X con animación gráfica (para mayor información, consulte la sección **Tipos de ventanas (Compatibilidad)**). Por defecto, si esta constante no se utiliza, el comando mostrará una caja de diálogo estándar.

Si no quiere utilizar una opción, pase 0 en el parámetro *opciones*.

El parámetro opcional *seleccionados* le permite obtener la ruta de acceso completa (ruta de acceso + nombre) de cada archivo seleccionado por el usuario. El comando crea, dimensiona y llena el array de acuerdo a la selección del usuario. Este parámetro es útil cuando la opción Multiple files se utiliza o cuando quiere conocer la ruta de acceso del archivo seleccionado (simplemente tome el nombre del archivo devuelto por el comando del valor del array). Si ningún archivo está seleccionado, el array se devuelve vacío.

Nota: bajo Mac OS, un paquete seleccionado se considera como una carpeta. La ruta de acceso devuelta en el array *seleccionados* incluye un carácter final ":" . Por ejemplo: **Disk:Applications:4D:4D v11.4:US:4D Volume Desktop.app:**

El comando devuelve el nombre (nombre + extensión bajo Windows) del archivo seleccionado. Si son seleccionados varios archivos, el comando devuelve el nombre del primer archivo de la lista de archivos seleccionados. La lista de archivos puede recuperarse en el parámetro *seleccionados*. Si ningún archivo es seleccionado, el comando devuelve una cadena vacía.

Ejemplo 1

Este ejemplo se utiliza para especificar un archivo de datos 4D:

```
C_LONGINT($plataforma)
PLATFORM PROPERTIES($plataforma)
If($plataforma=Windows)
   $DocTipo:=".4DD"
Else
   $DocTipo:="com.4d.4d.data-file"  `UTI type
End if
$Opciones:=Alias selection+Package open+Use sheet window
$Doc:=Select document("";$DocTipo;"Seleccionar el archivo de datos";$Opciones)
```

Ejemplo 2

Creación de un documento personalizado por el usuario:

```
$doc:=Select document(System folder(Documents folder)+"Report.pdf";"pdf";"Report name:";File name entry)
If(OK=1)
   BLOB TO DOCUMENT(Document;$blob)  // $blob contiene el documento a registrar
End if
```

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente y se seleccionó un documento válido, la variable sistema OK toma el valor 1 y la variable sistema Document contendrá la ruta de acceso completa del archivo seleccionado.

Si no se seleccionó ningún archivo (por ejemplo, si el usuario hace clic en el botón **Cancelar** en la caja de diálogo de apertura de archivos), la variable sistema OK toma el valor 0 y la variable Document estará vacía.

Select folder

Select folder ({mensaje }{;}{ rutaDefecto {; opciones}}) -> Resultado

Parámetro	Tipo		Descripción
mensaje	Cadena	➡	Título de la ventana
rutaDefecto	Cadena, Entero largo	➡	• Ruta de acceso por defecto o • Ruta vacía para mostrar el usuario por defecto carpeta ("Mis documentos" bajo Windows, "Documentos" bajor Mac OS), o • Número de ruta de acceso memorizada
opciones	Entero largo	➡	Opciones de selección bajo Mac OS
Resultado	Cadena	➡	Ruta de acceso al archivo seleccionado

Descripción

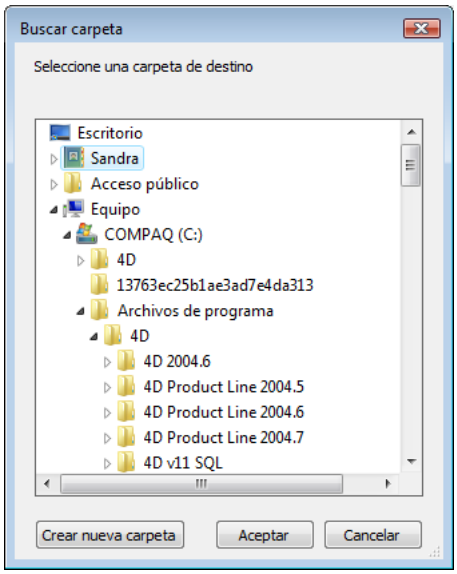
El comando **Select folder** muestra una caja de diálogo que le permite seleccionar manualmente una carpeta y luego recuperar la ruta de acceso completa a esa carpeta. El parámetro opcional *rutaDefecto* puede utilizarse para designar la ubicación de una carpeta que inicialmente será mostrada en la caja de diálogo de selección de la carpeta.

Nota: este comando no modifica la carpeta actual de la aplicación 4D.

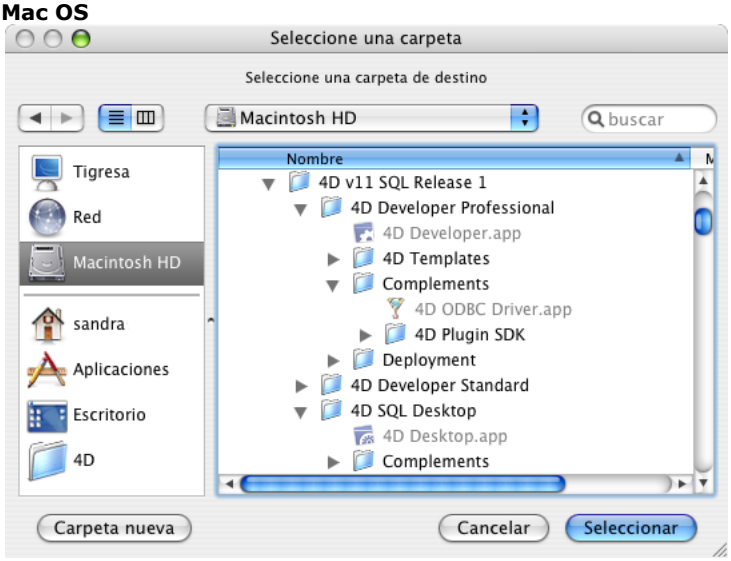
El comando **Select folder** muestra una caja de diálogo estándar de navegación a través de los volúmenes y las carpetas.

El parámetro opcional *mensaje* le permite mostrar un mensaje en la caja de diálogo. En los siguientes ejemplos, el mensaje es "Seleccione una carpeta de destino":

Windows



Mac OS



Puede utilizar el parámetro *rutaDefecto* para ofrecer una ubicación de carpeta por defecto en la caja de diálogo de selección de carpeta. Puede pasar en este parámetro tres tipos de valores:

- Una ruta de acceso de una carpeta válida utilizando la sintaxis de la plataforma actual.
- Una cadena vacía ("") para mostrar la carpeta del usuario por defecto del sistema ("Mis documentos" bajo Windows, "Documentos" bajo Mac OS).
- El número de ruta de acceso memorizada (de 1 a 32 000) para mostrar la carpeta asociada. Esto significa que puede almacenar en memoria la ruta de acceso de la carpeta abierta cuando el usuario hace clic en el botón de selección; en otras palabras, la carpeta elegida por el usuario. Cuando se llama un número arbitrario (por ejemplo, 5) el comando muestra la

carpeta del usuario por defecto del sistema (equivalente a pasar una cadena vacía). El usuario puede entonces navegar entre las carpetas en su disco duro. Cuando el usuario hace clic en el botón de selección, la ruta de acceso se memoriza y se asocia con el número 5. Cuando posteriormente se llama el número 5, se utilizará por defecto la ruta de acceso memorizada. En caso de selección de una nueva ubicación, la ruta número 5 se actualizará, y así sucesivamente.
Este mecanismo puede utilizarse para memorizar hasta 32 000 rutas de acceso. Bajo Windows, cada ruta se conserva durante la sesión. Bajo Mac OS, las rutas quedan memorizadas de una sesión a otra. Si la ruta es incorrecta, el parámetro *rutaDefecto* se ignora.

Nota: este mecanismo es idéntico al utilizado por el comando **Select document**. Los números de rutas de acceso memorizadas se comparten entre los dos comandos.

El parámetro *opciones* le permite beneficiarse de funciones adicionales bajo Mac OS. En este parámetro, puede pasar una de las siguientes constantes, del tema :

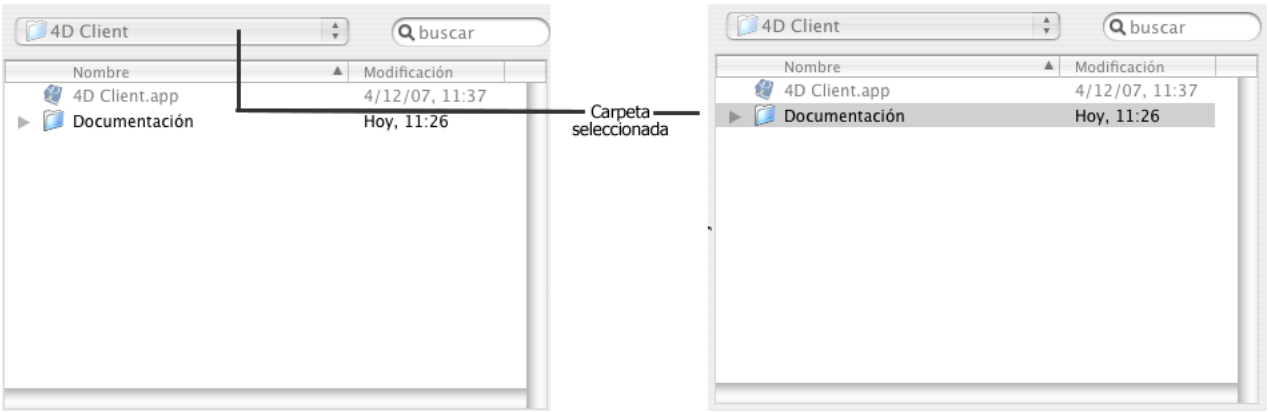
Constante	Tipo	Valor	Comentario
Package open	Entero largo	2	(Mac OS únicamente): autoriza la apertura de paquetes como carpetas y por lo tanto la visualización/selección de sus contenidos. Por defecto, si no se utiliza esta constante, el comando no permitirá la apertura de paquetes.
Use sheet window	Entero largo	16	(Mac OS únicamente): muestra la caja de diálogo de selección en forma de una ventana hoja (esta opción se ignora en Windows). Las ventanas hojas son específicas para la interfaz Mac OS X con animación gráfica (para mayor información, consulte la sección **DISPLAY SELECTION**). Por defecto, si esta constante no se utiliza, el comando mostrará una caja de diálogo estándar.

Puede pasar una constante o la combinación de dos. Estas opciones sólo son tenidas en cuenta bajo Mac OS. Bajo Windows, el parámetro *opciones* es ignorado si se pasa.

El usuario selecciona una carpeta y luego hace clic en el botón **Aceptar** (en Windows) o **Seleccionar** (en Mac OS). La ruta de acceso a la carpeta luego es devuelta por la función.

- En Windows, la ruta de acceso devuelta tiene el siguiente formato:
  "C:\Carpeta1\Carpeta2\CarpetaSeleccionada\"
- En Mac OS, la ruta de acceso devuelta tiene el siguiente formato:
  "Disco:Carpeta1:Carpeta2:CarpetaSeleccionada:"

Nota: bajo Mac OS, dependiendo de si el nombre de la carpeta se selecciona o no en la caja de diálogo, la ruta de acceso devuelta podría ser diferente.



4D Server: esta función permite visualizar los volúmenes conectados a las estaciones de trabajo de los clientes. No es posible llamar esta función desde un procedimiento almacenado.

Si el usuario valida la caja de diálogo, la variable sistema **OK** toma el valor 1. Si el usuario hace clic en el botón **Cancelar**, la variable sistema **OK** toma el valor 0 y la función devuelve una cadena vacía.

Nota: en Windows, si el usuario selecciona algunos elementos incorrectos, tales como "Puesto de trabajo", "Papelera", etc., la variable sistema **OK** toma el valor 0, incluso si el usuario valida la caja de diálogo.

Ejemplo

El siguiente ejemplo le permite seleccionar la carpeta en la cual se almacenarán las imágenes de la librería de imágenes:

```
$PictFolder:=Select folder("Seleccione una carpeta para sus imágenes")
PICTURE LIBRARY LIST(pictRefs;pictNames)
For($n;1;Size of array(pictNames))
   GET PICTURE FROM LIBRARY(pictRefs{$n};$vStoredPict)
   WRITE PICTURE FILE($PictFolder+pictNames{$n};$vStoredPict)
End for
```

SET DOCUMENT POSITION

SET DOCUMENT POSITION (docRef ; offset {; ancla})

Parámetro	Tipo		Descripción
docRef	DocRef	➡	Número de referencia del documento
offset	Real	➡	Posición del archivo (expresada en bytes)
ancla	Entero largo	➡	1 = En relación con el inicio del archivo 2 = En relación con el final del archivo 3 = En relación con la posición actual

Descripción

Este comando funciona únicamente en un documento abierto cuyo número de referencia se pasa en *docRef*.

SET DOCUMENT POSITION define la posición que se pasa en *offset* donde ocurrirá la próxima lectura (**RECEIVE PACKET**) o escritura (**SEND PACKET**).

Si omite el parámetro opcional *ancla*, la posición es relativa al inicio del documento. Si especifica el parámetro *ancla*, pase uno de los valores listados anteriormente.

Dependiendo del ancla puede pasar valores positivos o negativos en *offset*.

SET DOCUMENT PROPERTIES

SET DOCUMENT PROPERTIES (doc ; bloqueado ; invisible ; creado el ; creado a las ; modificado el ; modificado a las)

Parámetro	Tipo		Descripción
doc	Cadena	➡	Nombre del documento o ruta de acceso completa al documento
bloqueado	Booleano	➡	Bloqueado (True) o desbloqueado (False)
invisible	Booleano	➡	Invisible (True) o Visible (False)
creado el	Fecha	➡	Fecha de creación
creado a las	Hora	➡	Hora de creación
modificado el	Fecha	➡	Última fecha de modificación
modificado a las	Hora	➡	Hora de la última modificación

Descripción

El comando **SET DOCUMENT PROPERTIES** modifica la información del documento cuyo nombre o ruta de acceso se pasa en *documento*.

Antes de llamar:

- Pase True en *bloqueado* para bloquear el documento. Un documento bloqueado no puede ser modificado. Pase False en *bloqueado* para desbloquear un documento.
- Pase True en *invisible* para ocultar el documento. Pase False en *invisible* para volver visible el documento en las ventanas del escritorio.
- Pase la fecha y hora de creación del documento en *creado el* y *creado a las*.
- Pase la fecha y hora de la última modificación del documento en *modificado en* y *modificado a las*.

Las fechas y horas de creación y última modificación son administradas por el administrador de archivos de su sistema cada vez que crea o accede a un documento. Utilizando este comando, puede cambiar estas propiedades en casos particulares. Ver el ejemplo del comando **GET DOCUMENT PROPERTIES**.

SET DOCUMENT SIZE

SET DOCUMENT SIZE (docRef ; tamaño)

Parámetro	Tipo		Descripción
docRef	DocRef	➡	Número de referencia del documento
tamaño	Real	➡	Nuevo tamaño expresado en bytes

Descripción

El comando **SET DOCUMENT SIZE** define el tamaño de un documento para el número de bytes que se pasan en *tamaño*.
Si el documento está abierto, pase su número de referencia en *docRef*.
En Macintosh, se modifica el tamaño del data fork del documento.

SHOW ON DISK

SHOW ON DISK (nombreRuta {; *})

Parámetro	Tipo		Descripción
nombreRuta	Cadena	➡	Ruta de acceso del elemento a mostrar
*	Operador	➡	Si el elemento es una carpeta, mostrar su contenido

Descripción

El comando **SHOW ON DISK** muestra en una ventana estándar del sistema operativo el archivo o la carpeta en la cual la ruta de acceso se pasa en el parámetro *nombreRuta.*

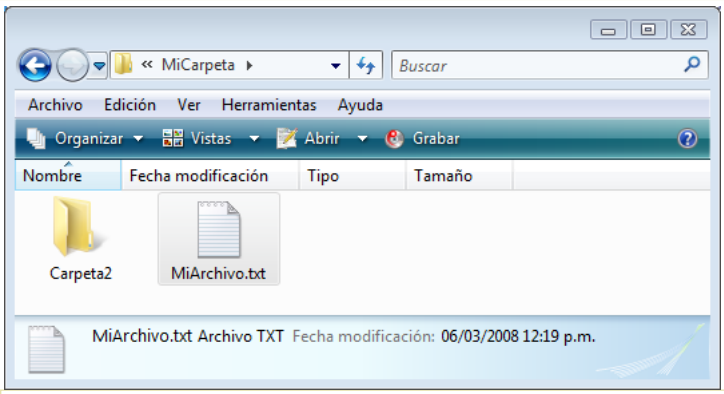
En una interfaz de usuario, este comando permite designar la ubicación de un archivo o carpeta especifico.

Por defecto, si *nombreRuta* designa una carpeta, el comando muestra el nivel de la carpeta misma. Si pasa el parámetro opcional *, el comando abre la carpeta y muestra su contenido en la ventana. Si *nombreRuta* designa un archivo, se ignora el parámetro *.

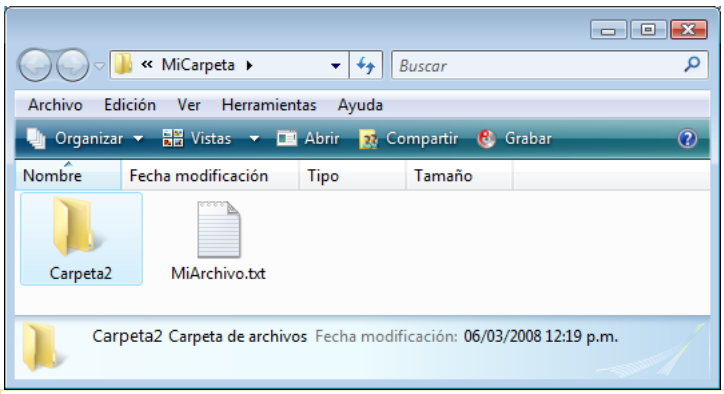
Ejemplo

Los siguientes ejemplos ilustran el funcionamiento del comando:

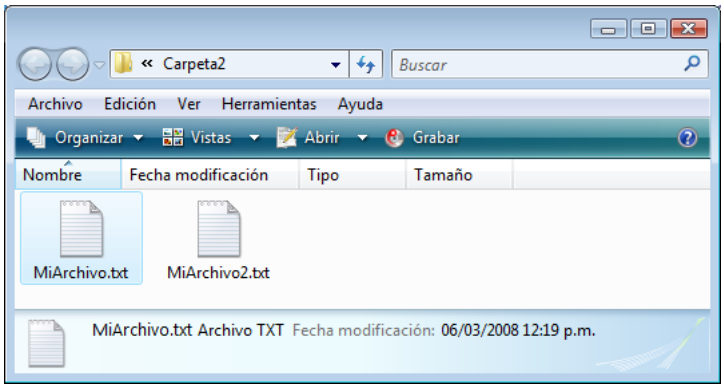
```
SHOW ON DISK("c:\\MiCarpeta\\MiArchivo.txt") ` Muestra el archivo designado
```




```
SHOW ON DISK("c:\\MiCarpeta\\Carpeta2") ` Muestra la carpeta designada
```




```
SHOW ON DISK("c:\\MiCarpeta\\Carpeta2";*) ` Muestra los contenidos de la carpeta designada
```



Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si el comando se ejecuta correctamente, de lo contrario toma el valor 0.

Test path name

Test path name (nombreRuta) -> Resultado

Parámetro	Tipo		Descripción
nombreRuta	Cadena	➡	Ruta de acceso a un directorio, carpeta o documento
Resultado	Entero largo	⮌	1, rutaAcceso es un documento existente 0, rutaAcceso es un directorio o carpeta existente <0, ruta de acceso incorrecta, código de error del administrador de archivos del sistema

Descripción

La función **Test path name** verifica si un documento o carpeta cuyo nombre o ruta de acceso se pasa en *rutaAcceso* está presente en el disco. Puede pasar una ruta de acceso relativa o absoluta, expresada en la sintaxis del sistema actual.

Si se encuentra un documento, **Test path name** devuelve 1. Si se encuentra una carpeta, **Test path name** devuelve 0.

4D ofrece las siguientes constantes predefinidas:

Constante	Tipo	Valor
Is a document	Entero largo	1
Is a folder	Entero largo	0

Si no se encuentra ningún documento o carpeta, **Test path name** devuelve un valor negativo (por ejemplo -43 para "Archivo no encontrado").

Ejemplo

El siguiente ejemplo prueba la presencia del documento "Diario" en la carpeta de la base, si no lo encuentra lo crea:

```
If(Test path name("Diario")#Is a document)
    $vhDocRef:=Create document("Diario")
    If(OK=1)
        CLOSE DOCUMENT($vhDocRef)
    End if
End if
```

TEXT TO DOCUMENT

TEXT TO DOCUMENT (nomArchivo ; texto {; conjCaract {; ModoRetorno}})

Parámetro	Tipo		Descripción
nomArchivo	Cadena	➡	Nombre del documento o ruta de acceso al documento
texto	Texto	➡	Texto a almacenar en el documento
conjCaract	Texto, Entero largo	➡	Nombre o número del conjunto de caracteres
ModoRetorno	Entero largo	➡	Modo de procesamiento para los retornos de línea

Descripción

El comando **TEXT TO DOCUMENT** le permite escribir el *texto* directamente en el archivo de disco.

En *nomArchivo*, pase el nombre o ruta de acceso del archivo a escribir. Si el archivo no existe, se crea. Cuando el archivo ya existe en el disco, su contenido anterior se borrará, excepto si ya está abierto, en cuyo caso, su contenido se bloquea y se genera un error. En *nomArchivo*, que puede pasar:

- sólo el nombre del archivo, por ejemplo "miArchivo.txt": en este caso, el archivo se coloca junto al archivo de estructura de la aplicación.
- una ruta de acceso relativa al archivo de estructura de la aplicación, por ejemplo, "\\docs\\miArchivo.txt" en Windows o ":docs:miArchivo.txt" en OS X.
- una ruta de acceso absoluta, por ejemplo, "c:\\app\\docs\\miArchivo.txt" en Windows o "MacHD:docs:miArchivo.txt" en OS X.

Si desea que el usuario sea capaz de indicar el nombre o la ubicación del documento, utilice los comandos **Open document** o **Create document** , así como la variable del sistema **Document**.

Nota: por defecto, los documentos generados por este comando no tienen una extensión. Debe pasar una extensión en *nomArchivo*. También puede utilizar el comando **_o_SET DOCUMENT TYPE**.

En *texto*, pase el texto a escribir en el disco. Puede ser una constante literal ("mi texto"), o un campo o variable texto 4D.

En *conjCaract*, pase el conjunto de caracteres a utilizar para la escritura del documento. Puede pasar una cadena con el nombre estándar del conjunto (por ejemplo, "ISO-8859-1" o "UTF-8") o su ID MIBEnum (entero largo). Para más información sobre la lista de conjuntos de caracteres soportados por 4D, consulte la descripción del comando **CONVERT FROM TEXT**. Si un Byte Order Mark (BOM) existe para el conjunto de caracteres, 4D lo inserta en el documento. Si no se especifica un conjunto de caracteres, 4D utiliza por defecto el conjunto de caracteres "UTF_8 " y un BOM.

En *modoRetorno*, puede pasar un entero largo que indica el proceso a aplicar a los caracteres de fin de línea antes de guardarlos en el archivo. Puede pasar una de las siguientes constantes, ubicadas en el tema "**Documentos sistema**":

Constante	Tipo	Valor	Comentario
Document unchanged	Entero largo	0	Ningún proceso
Document with CR	Entero largo	3	Las líneas de ruptura se convierten al formato OS X: CR (*retorno de carro*)
Document with CRLF	Entero largo	2	Las líneas de ruptura se convierten al formato Windows: CRLF (*retorno de carro* + *salto de línea*)
Document with LF	Entero largo	4	Las líneas de ruptura se convierten al formato Unix: LF (*salto de línea*)
Document with native format	Entero largo	1	(Por defecto) las líneas de ruptura se convierten al formato nativo del sistema operativo: CR (retorno de carro) en OS X, CRLF (*retorno de carro* + *salto de línea*) en Windows

Por defecto, si omite el parámetro *modoRetorno*, los caracteres de fin de línea se procesan en modo nativo (1).

Nota: este comando no modifica la variable OK. En caso de falla, se genera un error que puede interceptar utilizando un método instalado por el comando **ON ERR CALL**.

Ejemplo 1

Estos son algunos ejemplos típicos de uso de este comando:

```
TEXT TO DOCUMENT("myTest.txt";"Esta es una prueba")
TEXT TO DOCUMENT("myTest.xml";"Esta es una prueba")
```

Ejemplo 2

Ejemplo que permite al usuario indicar la ubicación del archivo a crear:

```
$MyTextVar:="Esta es una prueba"
ON ERR CALL("IO ERROR HANDLER")
$vhDocRef :=Create document("")
  // Guardar documento con la extensión ".txt"
  // En este caso, la extensión .txt siempre se añade al nombre; no es posible cambiarla
```

```
If(OK=1)  // Si el documento se ha creado con éxito
    CLOSE DOCUMENT($vhDocRef)  //Cerrar el documento
    TEXT TO DOCUMENT(Document;$MyTextVar )
 // Escribimos el documento
Else
 // Gestión de error
End if
```

VOLUME ATTRIBUTES

VOLUME ATTRIBUTES (volumen ; tamaño ; utilizado ; libre)

Parámetro	Tipo		Descripción
volumen	Cadena	➡	Nombre del volumen
tamaño	Real	⬅	Tamaño del volumen expresado en bytes
utilizado	Real	⬅	Espacio utilizado expresado en bytes
libre	Real	⬅	Espacio libre expresado en bytes

Descripción

El comando **VOLUME ATTRIBUTES** devuelve en bytes el tamaño, el espacio utilizado y el espacio libre del volumen cuyo nombre se pasa en *volumen*.

Nota: si *volumen* indica un volumen remoto no montado, la variable OK toma el valor 0 y los tres parámetros devuelven -1.

Ejemplo

Su aplicación incluye algunas operaciones por lotes que se ejecutan en la noche o los fines de semana para almacenar archivos temporales grandes en disco. Para que este proceso sea tan automático y flexible como sea posible, usted escribe una rutina que buscará automáticamente el primer volumen cuyo espacio libre sea suficiente para sus archivos temporales. He aquí el método:

```
  ` Método de proyecto Buscar volumen para espacio
  ` Buscar volumen para espacio ( Real ) -> Alfa
  ` Buscar volumen para espacio ( Espacio necesario en bytes ) -> Nombre del volumen o cadena vacía

C_STRING(31;$0)
C_STRING(255;$vsDocNombre)
C_LONGINT($vlNbVolumenes;$vlVolumenes)
C_REAL($1;$vlTamaño;$vlUtilizado;$vlLibre)
C_TIME($vhDocRef)

  ` Inicializar el resultado de la función
$0:=""
  ` Proteger todas las operaciones de entrada/salida con un método de interrupción de errores
ON ERR CALL("ERROR METHOD")
  ` Obtener la lista de los volúmenes
ARRAY STRING(31;$asVolumenes;0)
gError:=0
VOLUME LIST($asVolumenes)
If(gError=0)
  ` Si se corre en Windows, ignorar los dos lectores de diskettes
   If(On Windows)
      $vlVolumen:=Find in array($asVolumenes;"A:\\")
      If($vlVolumen>0)
         DELETE FROM ARRAY($asVolumenes;$vlVolumen)
      End if
      $vlVolumen:=Find in array($asVolumenes;"B:\\")
      If($vlVolumen>0)
         DELETE FORM ARRAY($asVolumenes;$vlVolumen)
      End if
   End if
   $vlNbVolumenes:=Size of array($asVolumenes)
  ` Para cada volumen
   For($vlVolumen;1;$vlNbVolumenes)
  ` Obtener el tamaño, el espacio utilizado y el espacio libre
      gError:=0
      VOLUME ATTRIBUTES($asVolumenes{$vlVolumen};$vlTamaño;$vlUtilizado;$vlLibre)
      If(gError=0)
  ` ¿El espacio libre es suficiente (más de 32K extra) ?
         If($vlLibre>=($1+32768))
  ` Si es así, verificar si el volumen no está bloqueado..
            $vsDocNombre:=$asVolumenes{$vlVolumen}+Char(Directory symbol)+"XYZ"+String(Random)+".TXT"
            $vhDocRef:=Create document($vsDocNombre)
            If(OK=1)
               CLOSE DOCUMENT($vhDocRef)
               DELETE DOCUMENT($vsDocNombre)
  ` Si todo está bien, devolver el nombre del volumen
```

```
                $0:=$asVolumenes{$vlVolumen}
                $vlVolumen:=$vlNbVolumenes+1
            End if
        End if
    End if
  End for
End if
ON ERR CALL("")
```

Una vez se añade este método de proyecto a su aplicación, puede escribir:

```
$vsVolumen:=Buscar volumen para espacio(100*1024*1024)
If($vsVolumen#"")
  ` Continuar
Else
  ALERT("¡Es necesario un volumen con por lo menos 100 MB de espacio libre!")
End if
```

VOLUME LIST

VOLUME LIST (volumenes)

Parámetro	Tipo		Descripción
volumenes	Array cadena	⇐	Nombres de los volúmenes montados actualmente

Descripción

El comando **VOLUME LIST** llena el array *volumenes*, de tipo texto, con los nombres de los volúmenes definidos (Windows) o montados (Mac OS) en su equipo.

- En Macintosh, devuelve la lista de volúmenes visibles al nivel del Finder. Sólo se devuelven los nombres de los volúmenes (por ejemplo "MacHD", "BootCamp", etc.).
- En Windows, devuelve la lista de los volúmenes actualmente definidos sin importar si el volumen está presente físicamente presente (por ejemplo el volumen **E:** se devolverá sin importar si hay un CD o DVD en el drive). Los nombres de los volúmenes están seguidos por el carácter separador de carpetas ("C:\").

Ejemplo

Utilizando un área de desplazamiento llamada *atVolumenes,* usted quiere mostrar la lista de volúmenes definidos o montados en su equipo, entonces escribe:

```
Case of
   :(Form event=On Load)
      ARRAY TEXT(atVolumenes;0)
      VOLUME LIST(atVolumenes)

 //...
End case
```

_o_Document creator

_o_Document creator (doc) -> Resultado

Parámetro	Tipo		Descripción
doc	Cadena	➡	Nombre del documento o Ruta de acceso completa a un documento
Resultado	Cadena	⮌	Cadena vacía (Windows) o Creador del archivo (Mac OS)

Nota de compatibilidad

Este comando está en desuso. No hace nada en 4D v16 R6 y superior.

_o_Document type

_o_Document type (doc) -> Resultado

Parámetro	Tipo		Descripción
doc	Cadena	⇒	Nombre del documento
Resultado	Cadena	⇐	Extensión de archivo Windows (cadena de 1 a 3 caracteres) o tipo de archivo Mac OS (cadena de 4 caracteres)

Nota de compatibilidad

Este comando es obsoleto. Utilice el comando **Path to object** para extraer la extensión del archivo.

_o_MAP FILE TYPES

_o_MAP FILE TYPES (macOS ; windows ; contexto)

Parámetro	Tipo		Descripción
macOS	Cadena	➡	Tipo de archivo Mac OS (4 caracteres)
windows	Cadena	➡	Extensión de archivo Windows
contexto	Cadena	➡	Cadena mostrada en la lista desplegable de los tipos de archivos en la caja de diálogo de apertura de archivos en Windows

Nota de compatibilidad

A partir de 4D v16 R6, este comando es obsoleto y ya no se debe utilizar.

El uso de tipos de archivos de Mac OS ha quedado obsoleto para muchas versiones de macOS. Al igual que Windows, la identificación de archivos se basa en las extensiones de nombre. Puede utilizar los comandos **Object to path** y **Path to object** para administrar las extensiones de archivos. También puede usar las UTIs junto con las declaraciones de Info.plist para manejar los tipos de archivos.

_o_SET DOCUMENT CREATOR

_o_SET DOCUMENT CREATOR (doc ; creador)

Parámetro	Tipo		Descripción
doc	Cadena	⇒	Nombre del documento o Ruta de acceso completa a un documento
creador	Cadena	⇒	Creador del archivo (Mac OS) o cadena vacía (Windows)

Nota de compatibilidad

Este comando está en desuso. No hace nada en 4D v16 R6 y superior.

_o_SET DOCUMENT TYPE

_o_SET DOCUMENT TYPE (doc ; tipo)

Parámetro	Tipo		Descripción
doc	Cadena	➡	Nombre del documento o ruta de acceso completa a un documento
tipo	Cadena	➡	Extensión de archivo Windows o tipo de archivo Mac OS (cadena de 4 caracteres)

Nota de compatibilidad

Este comando está en desuso en 4D v16 R6 y superior. Ahora puede utilizar el comando **Object to path** para modificar la extensión del archivo.

Entorno 4D

- Application file
- Application type
- Application version
- BUILD APPLICATION
- CHANGE LICENSES
- Compact data file
- COMPONENT LIST
- CREATE DATA FILE
- Data file
- Get 4D file
- Get 4D folder
- Get database localization
- Get database measures
- Get database parameter
- Get last update log path
- Get license info
- GET SERIAL INFORMATION
- Get table fragmentation
- Is compiled mode
- Is data file locked
- Is license available
- NOTIFY RESOURCES FOLDER MODIFICATION
- OPEN ADMINISTRATION WINDOW
- OPEN DATA FILE
- OPEN DATABASE
- OPEN SECURITY CENTER
- OPEN SETTINGS WINDOW
- PLUGIN LIST
- QUIT 4D
- RESTART 4D
- SET DATABASE LOCALIZATION
- SET DATABASE PARAMETER Updated 17.0
- SET UPDATE FOLDER
- Structure file
- VERIFY CURRENT DATA FILE
- VERIFY DATA FILE
- Version type
- *_o_ADD DATA SEGMENT*
- *_o_DATA SEGMENT LIST*

Application file

Application file -> Resultado

Parámetro	Tipo		Descripción
Resultado	Cadena		Nombre largo del archivo 4D ejecutable o de la aplicación 4D

Descripción

El comando **Application file** devuelve el nombre largo del archivo ejecutable o de la aplicación 4D que está utilizando.

En Windows
Si está utilizando por ejemplo 4D ubicado en \PROGRAMS\4D en el disco E, el comando devuelve E:\PROGRAMS\4D\4D.exe.

En Macintosh
Si está corriendo por ejemplo 4D en la carpeta Programas del disco Macintosh HD, el comando devuelve Macintosh HD:Programs:4D.app.

Ejemplo

Al iniciar su base de datos en Windows, necesita verificar si una librería DLL se encuentra ubicada en el mismo nivel del archivo ejecutable 4D. En el método de base de datos **On Startup** de su aplicación puede escribir:

```
If(En Windows  &  (Application type#4D Server))
   If(Test path name(Extraer_ruta_de_acceso(Application file)+"XRAYCAPT.DLL")#Is a document)
  ` Mostrar una caja de diálogo explicando que la librería XRAYCAPT.DLL
  ` no está. Por lo tanto, la captura de rayos X no estará disponible.
   End if
End if
```

Nota: los métodos de proyecto **En_Windows** y **Extraer_ruta_de_acceso** se detallan en la sección **Documentos del sistema**.

Application type

Application type -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Valor numérico representando el tipo de la aplicación

Descripción

El comando **Application type** devuelve un valor numérico que representa el tipo de entorno 4D que usted está utilizando. 4D ofrece las siguientes constantes predefinidas:

Constante	Tipo	Valor
4D Desktop	Entero largo	3
4D Local mode	Entero largo	0
4D Remote mode	Entero largo	4
4D Server	Entero largo	5
4D Volume desktop	Entero largo	1

Nota: *4D Desktop* incorpora ciertas ofertas de despliegue, como por ejemplo,"4D SQL Desktop".

Ejemplo

En alguna parte de su código, diferente del método de base de datos **On Server Startup**, necesita verificar si está corriendo 4D Server. Puede escribir:

```
If(Application type=4D Server)
   ` Realizar las acciones apropiadas
End if
```

Application version

Application version {(buildNum {; *})} -> Resultado

Parámetro	Tipo		Descripción
buildNum	Entero largo	⇐	Número de build
*	Operador	⇒	Si pasa número de versión largo, de lo contrario número de versión corto
Resultado	Cadena	⇒	Número de versión en una cadena codificada

Descripción

El comando **Application version** devuelve una cadena codificada que expresa el número de versión del entorno 4D que está utilizando.

- Si no pasa el parámetro opcional *, devuelve una cadena de 4 caracteres, con el siguiente formato:

Caracteres	Descripción
1-2	Número de versión
3	Número "R"
4	Número de revisión

- Si pasa el parámetro opcional *, devuelve una cadena de 8 caracteres, con el siguiente formato:

Caracteres	Descripción
1	"F" representa una versión final
	"B" representa una versión beta
	Los otros caracteres representan una versión interna de 4D
2-3-4	Número de compilación interno de 4D
5-6	Número de versión
7	Número de "R"
8	Número de revisión

Nota de compatibilidad (4D v14)

La numeración de versiones ha cambiado desde la versión 14 de 4D:

- el **número "R"** es el número de versión "R" de 4D, por ejemplo 3 para la versión R3 (contiene 0 para una versión "bug fix"),
- el **número de revisión** es el número de versión "bug fix" de 4D (contiene 0 para una versión "R").

En las versiones anteriores de 4D, el número de versión "R" era el número de actualización, designaba la revisión y el número de revisión era siempre 0.

Ejemplos para un número de versión corto:

Versiones	Valor devuelto	
4D v13.1	"1310"	*Sistema de numeración anterior*
4D v14 R2	"1420"	Release R2
4D v14 R3	"1430"	Release R3
4D v14.1	"1401"	Primera versión "bug fix" de 4D v14
4D v14.2	"1402"	Segunda versión "bug fix" de 4D v14

Ejemplos para un número de versión largo:

Versiones	Valor devuelto
4D v12.5 beta	"B0011250"
4D v14 R2 beta	"B0011420"
4D v14 R3 final	"F0011430"
4D v14.1 beta	"B0011401"

El comando **Application version** puede devolver información adicional en el parámetro opcional *numBuild*: el número de build de la versión actual de la aplicación 4D. Este es un número de compilación interno que se puede utilizar para el control de versiones, o al contactar al departamento de Servicios Técnicos de 4D.

Nota: en el caso de las aplicaciones compiladas y fusionadas con 4D Volume License, el número de build devuelto no es significativo. En este contexto, la información de versión es administrada por el desarrollador.

Ejemplo 1

Este ejemplo muestra el número de versión del entorno 4D:

```
$vs4Dversion:=Application version
ALERT("Usted está utilizando la versión "+String(Num(Substring($vs4Dversion;1;2)))+"."+
$vs4Dversion[[3]]+"."+$vs4Dversion[[4]])
```

Ejemplo 2

Este ejemplo hace una prueba para verificar si usted está utilizando una versión final:

```
If(Substring(Application version(*);1;1)#"F")
   ALERT("Por favor asegúrese de utilizar una versión final de 4D con esta base")
   QUIT 4D
End if
```

Ejemplo 3

Usted desea utilizar el valor de la versión corta de la aplicación devuelto por el comando para mostrar el nombre lanzamiento de la aplicación 4D. Puede escribir:

```
C_LONGINT($Lon_build)
C_TEXT($Txt_info;$Txt_major;$Txt_minor;$Txt_release;$Txt_version)

$Txt_version:=Application version($Lon_build)

$Txt_major:=$Txt_version[[1]]+$Txt_version[[2]] //número de versión, por ejemplo 14
$Txt_release:=$Txt_version[[3]] //Rx
$Txt_minor:=$Txt_version[[4]] //.x

$Txt_info:="4D v"+$Txt_major
If($Txt_release="0") //4D v14.x
   $Txt_info:=$Txt_info+Choose($Txt_minor#"0";"."+$Txt_minor;"")

Else //4D v14 Rx
   $Txt_info:=$Txt_info+" R"+$Txt_release
End if
```

BUILD APPLICATION

BUILD APPLICATION {{ nomProyecto)}

Parámetro	Tipo		Descripción
nomProyecto	Cadena	➡	Ruta de acceso completa del proyecto a utilizar

Descripción

El comando **BUILD APPLICATION** lanza el proceso de generación de la aplicación teniendo en cuenta los parámetros definidos en el proyecto de aplicación actual o en el proyecto de aplicación designado por el parámetro *nomProyecto*.

Un proyecto de aplicación es un archivo XML que contiene todos los parámetros utilizados para generar una aplicación. La mayoría de estos parámetros son visibles en la caja de diálogo Generar aplicación... (Para mayor información, consulte la sección **El generador de aplicaciones** en el Manual de Diseño de 4D).

Por defecto, 4D crea para cada base un proyecto de aplicación llamado "buildapp.xml" (por defecto) para cada base y lo ubica en la subcarpeta BuildApp en la carpeta Preferencias de la base.

Si la base no ha sido compilada aún o si el código compilado está desactualizado, el comando lanza primero el proceso de compilación. En este caso, no aparece la ventana del compilador (a menos que ocurra un error), sólo se muestra una barra de progreso.

Puede ocultar esta barra de progreso utilizando el comando **MESSAGES OFF**.

Si no pasa el parámetro opcional *nomProyecto*, el comando muestra una caja de diálogo estándar de apertura de archivos, de manera que pueda designar un archivo de proyecto. Cuando la caja de diálogo se valida, la variable sistema Document contiene la ruta de acceso completa del archivo seleccionado. Si pasa la ruta de acceso y el nombre de un archivo XML de proyecto de aplicación válido (codificación UTF-8 y extensión ".xml"), el comando utilizará los parámetros definidos en el archivo. Para mayor información sobre la estructura y las llaves utilizables en un archivo XML de proyecto de aplicación, consulte el manual **4D Llaves XML BuildApplication**.

Ejemplo

Este ejemplo crea dos aplicaciones en un solo método:

```
BUILD APPLICATION("c:\\folder\\projects\\miproyecto1.xml")
If(OK=1)
   BUILD APPLICATION("c:\\folder\\projects\\miproyecto2.xml")
End if
```

Variables y conjuntos sistema

La variable sistema OK toma el valor 1 si el comando se ha ejecutado correctamente. De lo contrario, toma el valor 0. La variable sistema Document contiene la ruta de acceso completa al archivo de proyecto abierto.

Gestión de errores

Si el comando falla, se genera un error que puede interceptar con la ayuda del comando **ON ERR CALL**.

CHANGE LICENSES

CHANGE LICENSES
Este comando no requiere parámetros

Descripción

El comando **CHANGE LICENSES** muestra la caja de diálogo de gestión de licencias 4D.

Este comando sólo puede ser utilizado con aplicaciones 4D monousuario y no puede llamarse desde un componente. Cuando las contraseñas están habilitadas, este comando sólo puede ser ejecutado por el Diseñador o Administrador; no hace nada cuando es llamado por los usuarios que no tienen los derechos de acceso adecuados.

La caja de diálogo de gestión de licencias permite a un usuario activar los plug-ins o el servidor web en la máquina dónde se ejecuta. En 4D, puede mostrar esta caja de diálogo seleccionando el comando **Gestión de licencias.** en el menú **Ayuda**.

CHANGE LICENSES es una forma conveniente de activar licencia y añadir números de expansión en una aplicación 4D monousuario distribuida a los clientes. Los desarrolladores 4D y los administradores de sistemas pueden utilizar este comando para distribuir una aplicación 4D y permitir a los usuarios introducir su Licencia sin enviar una actualización a la aplicación cada vez.

Para mayor información esta caja de diálogo, consulte la sección **Instalación y activación** Guía de instalación de 4D.

Ejemplo

En una caja de diálogo de configuración o de preferencias personalizada, coloque un botón con el siguiente método:

```
// Método de objeto del botón bLicencia
CHANGE LICENSES
```

De esta forma un usuario puede activar licencias sin tener que modificar la base.

Compact data file

Compact data file (rutaEstructura ; rutaDatos {; carpetaArchivo {; opcion {; metodo}}}) -> Resultado

Parámetro	Tipo		Descripción
rutaEstructura	Texto	➡	Ruta de acceso al archivo de estructura
rutaDatos	Texto	➡	Ruta de acceso al archivo de datos
carpetaArchivo	Texto	➡	Ruta de acceso a la carpeta donde se colocará el archivo de datos original
opcion	Entero largo	➡	Opciones de compactación
metodo	Texto	➡	Nombre del método 4D de retrollamada
Resultado	Texto	↩	Ruta de acceso completa de la carpeta que contiene el archivo de datos original

Descripción

El comando **Compact data file** compacta el archivo de datos designado por el parámetro *rutaDatos* asociado al archivo de estructura *rutaEstructura*. Para mayor información sobre la compactación, consulte el Manual de Diseño.

Para asegurar la continuidad del funcionamiento de la base, el nuevo archivo de datos compactado reemplaza automáticamente al archivo original. Por seguridad, el archivo original no se modifica y se mueve a una carpeta especial llamada "Replaced files (compacting) YYYYMM-DD HH-MM-SS" donde YYYY-MM-DD HH-MM-SS representa la fecha y hora del backup. Por ejemplo: "Replaced files (compacting) 2007-09-27 15-20-35".

El comando devuelve la ruta de acceso completa de la carpeta efectivamente creada para almacenar el archivo de datos original. Este comando sólo puede ser ejecutado desde 4D (modo local) o 4D Server (procedimiento almacenado). El archivo de datos a compactar debe corresponder al archivo de estructura designado por *rutaEstructura*. Además, el archivo de datos no debe abrirse durante la ejecución del comando; de lo contrario se genera un error.

Si se produce un error durante el proceso de compactación, los archivos originales se conservan en su ubicación inicial. Si un archivo de índice (.4DIndx file) está asociado con el archivo de datos, también se compacta. Como para el archivo de datos, el archivo original se guarda y la nueva versión reemplaza la anterior.

- En el parámetro *rutaEstructura*, pase el nombre de la ruta completa del archivo de estructura asociado con el archivo de datos que quiere compactar. Esta información es necesaria para el proceso de compactación. La ruta de acceso debe expresarse en la sintaxis del sistema operativo. Igualmente puede pasar una cadena vacía; en este caso, aparece la caja de diálogo estándar de apertura de archivos de manera que pueda designar el archivo de estructura a utilizar.
- En el parámetro *rutaDatos*, puede pasar una cadena vacía, un nombre de archivo o una ruta completa de acceso, expresada en la sintaxis del sistema operativo. Si pasa una cadena vacía, aparece la caja de diálogo de apertura de archivos estándar de manera que el usuario pueda designar el archivo de datos a compactar. Este archivo debe corresponder al archivo de estructura definido en el parámetro *rutaEstructura*. Si pasa únicamente un nombre de archivo de datos, 4D lo buscará al mismo nivel que el archivo de estructura.
- El parámetro opcional *carpetaArchivo* puede ser utilizado para especificar la ubicación de la carpeta "Replaced files (compacting) FechaHora" destinada a recibir las versiones originales de los archivos de datos así como los eventuales archivos del índice.
  El comando devuelve la ruta de acceso completa de la carpeta creada efectivamente.
  - Si omite este parámetro, los archivos originales son colocados automáticamente en una carpeta "Replaced files (compacting) FechaHora" que se crea junto al archivo de estructura.
  - Si pasa una cadena vacía, aparece una caja de diálogo estándar de apertura de archivos permitiéndole al usuario designar la ubicación de la carpeta a crear.
  - Si pasa una ruta de acceso (expresada en la sintaxis del sistema operativo), el comando creará la carpeta "Replaced files (compacting) FechaHora" a esta ubicación.

- El parámetro opcional *opciones* se utiliza para definir diferentes opciones de compactación. Para hacerlo, utilice las siguientes constantes, del tema "**Mantenimiento archivo de datos**". Puede pasar varias opciones combinándolas:

Constante	Tipo	Valor	Comentario
Compact address table	Entero largo	131072	Fuerza la reescritura de la tabla de direcciones de los registros (ralentiza la compactación). Note que en este caso, los números de registro se reescriben. Si sólo pasa esta opción, 4D activa automáticamente la opción "Actualizar registros".
Create process	Entero largo	32768	Cuando se pasa esta opción, la compactación será asincrónica y deberá administrar los resultados utilizando el método de retrollamada (ver a continuación). 4D no mostrará la barra de progreso (es posible hacerlo vía el método de retrollamada). La variable sistema OK toma el valor 1 si el proceso se ha lanzado correctamente y 0 en todos los otros casos. Cuando no se pasa esta opción, la variable OK toma el valor 1 si la compactación se realiza correctamente, de lo contrario 0.
Do not create log file	Entero largo	16384	Por lo general, este comando crea un archivo de historial en formato XML (consulte el final de la descripción del comando). Con esta opción, no se creará un archivo de historial.
Timestamp log file name	Entero largo	262144	Cuando esta opción se pasa, el nombre del archivo de historial generado contendrá la fecha y hora de su creación; como resultado, no reemplazará cualquier archivo de historial generado anteriormente. Por defecto, si no se pasa esta opción, los nombres de archivos de historial no son marcados con la fecha y hora y cada nuevo archivo generado sustituye al anterior.
Update records	Entero largo	65536	Fuerza la rescritura de todos los registros en función de la definición actual de los campos en la estructura

- El parámetro *metodo* se utiliza para designar un método de retrollamada que será llamado regularmente durante la compactación si se pasa la opción Create process. De lo contrario, el método de retrollamada nunca será llamado. Para mayor información sobre este método, por favor consulte la descripción del comando **VERIFY DATA FILE**.
  Si el método de retrollamada no existe en la base, se genera un error y la variable sistema OK toma el valor 0.

Por defecto, el comando **Compact data file** crea un archivo de historial en formato XML (si no ha pasado la opción Do not create log file, ver el parámetro *opciones*). Su nombre está basado en el archivo de estructura de la base actual y está ubicado en la carpeta **Logs** de esta base. Por ejemplo, para un archivo de estructura llamado "myDB.4db," el archivo de historial será llamado "myDB_Compact_Log.xml."

Si ha pasado la opción Timestamp el nombre del archivo de historial incluye la fecha y la hora de su creación en la forma "AAAA-MM-DD HH-MM-SS", lo que nos da, por ejemplo: "myDB_Compact_Log_2015-09-27 15-20-35.xml". Esto significa que cada nuevo archivo de historial no reemplaza al anterior, pero podría requerir acción manual posterior para eliminar archivos innecesarios. Independientemente de la opción seleccionada, tan pronto como se genera un archivo de historial, su trayectoria se devuelve en la variable sistema Document después de la ejecución del comando.

Ejemplo

El siguiente ejemplo (Windows) efectúa la compactación de un archivo de datos:

```
$archivoEstructura:=Structure file
$archivoDatos:="C:\Bases\Facturas\Enero\Facturas.4dd"
$archivoOrig:="C:\Bases\Facturas\Archivos\Enero\"
$carpetaArch:=Compact data file($archivoEstructura;$archivoDatos;$archivoOrig)
```

Variables y conjuntos del sistema

Si la operación de compactación se lleva a cabo correctamente, la variable sistema OK toma el valor 1; de lo contrario, toma el valor 0. Sin un archivo de historial se ha generado, su ruta completa se devuelve en la variable sistema Document.

COMPONENT LIST

COMPONENT LIST (arrayComponentes)

Parámetro	Tipo		Descripción
arrayComponentes	Array texto	⇐	Nombres de los componentes

Descripción

El comando **COMPONENT LIST** dimensiona y llena el array *arrayComponentes* con los nombres de los componentes cargados por la aplicación 4D para la base local actual.

Al abrir una base, 4D carga los componentes válidos ubicados en la carpeta Componentes:

- la carpeta Componentes ubicada junto al archivo de estructura (si lo hay),
- la carpeta Componentes que se encuentra junto a la aplicación 4D ejecutable.

Recuerde: si el mismo componente está en las dos ubicaciones, 4D sólo cargará el ubicado junto a la estructura.

Este comando puede llamarse desde la base local o desde un componente. Si la base no utiliza componentes, el array *arrayComponentes* se devuelve vacío.

Los nombres de los componentes son los nombres de los archivos de estructura de las bases de las matrices (.4db, .4dc o .4dbase). Este comando puede utilizarse para configurar arquitecturas e interfaces modulares que ofrezcan funcionalidades adicionales de acuerdo a la presencia de los componentes.

Para mayor información sobre componentes 4D, por favor consulte el *Manual de Diseño*.

CREATE DATA FILE

CREATE DATA FILE (rutaAcceso)

Parámetro	Tipo		Descripción
rutaAcceso	Cadena	➡	Nombre o ruta de acceso completa del archivo de datos a abrir

Descripción

El comando **CREATE DATA FILE** permite crear un nuevo archivo de datos en disco y reemplazar rápidamente el archivo de datos abierto por la aplicación 4D.

El funcionamiento general de este comando es idéntico al del comando **OPEN DATA FILE**; la única diferencia es que el nuevo archivo de datos designado por el parámetro *rutaAcceso* se crea justo después de que se abre nuevamente la estructura.

Antes de lanzar la operación, el comando verifica que la ruta de acceso no corresponda a un archivo existente.

4D Server: a partir de 4D v13, este comando puede ejecutarse con 4D Server. En este contexto, efectúa una llamada interna a **QUIT 4D** en el servidor (lo que produce la aparición de una caja de diálogo en cada equipo remoto, indicando que el servidor está en proceso de salir) antes de la creación del archivo designado.

Data file

Data file {(segmento)} -> Resultado

Parámetro	Tipo		Descripción
segmento	Entero largo	➡	Obsoleto, no utilizar
Resultado	Cadena	⮌	Nombre largo del archivo de datos de la base

Descripción

El comando **Data file** devuelve el nombre largo del archivo de datos o del primer segmento de la base en la cual está trabajando actualmente.

A partir de la versión 11 de 4D, no se soportan segmentos de datos. El parámetro *segmento* se ignora y no debe utilizarse más.

En Windows
Si está trabajando por ejemplo con la base MisCDs en \DOCS\MisCDs en el disco G, una llamada a **Data file** devuelve G:\DOCS\MisCDs \MisCDs .4DD (siempre y cuando haya aceptado la ubicación y el nombre por defecto propuesto por 4D cuando creó la base).

En Macintosh
Si está trabajando por ejemplo con la base en la carpeta Documentos:MisCDs*ƒ*: en el disco Macintosh HD, una llamada a **Data file** devuelve Macintosh HD:Documentos:MisCDs*ƒ*:MisCDs*ƒ*.data (siempre y cuando haya aceptado la ubicación y el nombre por defecto propuesto por 4D cuando creó la base).

Advertencia: si llama este comando desde 4D en modo remoto, sólo se devuelve el nombre del archivo de datos, no el nombre largo.

Get 4D file

Get 4D file (archivo {; *}) -> Resultado

Parámetro	Tipo		Descripción
archivo	Entero largo	➡	Tipo de archivo
*	Operador	➡	Archivo devuelto de la base local
Resultado	Cadena	⬅	Ruta al archivo 4D

Descripción

El comando **Get 4D file** devuelve la ruta de acceso al archivo de entorno 4D especificado por el parámetro *archivo*. La ruta se devuelve utilizando la sintaxis del sistema.

Este comando le permite obtener la ruta de acceso real de archivos específicos, cuyo nombre o ubicación puede depender del contexto de la base. También le ayuda a escribir código genérico independiente de la versión 4D o el sistema operativo.

Pase en *archivo* un valor para definir el archivo que desea obtener el nombre de ruta completo. Puede utilizar una de las siguientes constantes, ubicadas en el tema "Entorno 4D":

Constante	Tipo	Valor	Comentario
Backup configuration file	Entero largo	1	Archivo Backup.xml, almacenado en la carpeta Preferencias/Backup junto al archivo de estructura de la base
Backup log file	Entero largo	13	Archivo de historial de la copia de seguridad actual. Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Build application log file	Entero largo	14	Archivo de historial actual en formato xml del generador de aplicaciones. Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Compacting log file	Entero largo	6	Archivo de historial de la compactación más reciente de la base, creada por el comando **Compact data file** o por el Centro de seguridad y mantenimiento (CSM). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Debug log file	Entero largo	12	Archivo de historial creado por el comando **SET DATABASE PARAMETER**(Debug log recording). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Diagnostic log file	Entero largo	11	Archivo de historial creado por **SET DATABASE PARAMETER**(Diagnostic log recording). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
HTTP debug log file	Entero largo	9	Archivo de historial creado por el comando **WEB SET OPTION**(Web debug log). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Last backup file	Entero largo	2	Último archivo de copia de seguridad, llamado <nombreBase>[bkpNum].4BK, almacenado en una ubicación personalizada
Repair log file	Entero largo	7	Archivo de historial de las reparaciones realizadas a la base por el Centro de mantenimiento y seguridad (CMS). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Request log file	Entero largo	10	Archivo de peticiones cliente/servidor estándar (excluyendo peticiones web) creado por los comandos **SET DATABASE PARAMETER**(4D Server log recording) o **SET DATABASE PARAMETER**(Client log recording). Si se ejecuta en el servidor, se devuelve el historial del servidor. Almacenado en la carpeta Logs en el servidor. Si se ejecuta en el cliente, se devuelve el historial del cliente. Almacenado en la carpeta Logs en el cliente. Si no existe ningún archivo de historial, se devuelve una ruta vacía.
User settings file	Entero largo	3	El archivo settings.4DSettings para los archivos de datos, almacenado en la carpeta Preferencias junto al archivo estructura de la base si se activa
User settings file for data	Entero largo	4	settings.4DSettings para el archivo de datos actual, almacenado en la carpeta Preferencias junto al archivo de datos.
Verification log file	Entero largo	5	Archivos de historial creados por los comandos **VERIFY CURRENT DATA FILE** y **VERIFY DATA FILE** o el Centro de mantenimiento y seguridad (CMS). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha realizado ninguna verificación o no existe ningún archivo de historial, se devuelve una ruta vacía. No se generan errores.
Web request log file	Entero largo	8	Archivo de historial creado por el comando **WEB SET OPTION**(Web log recording). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.

Cuando el comando se llama desde un componente, pase el parámetro opcional * para obtener la ruta del *archivo* de la base local. En este caso, si se omite el parámetro *, siempre se devuelve una cadena vacía.

En cuanto a User settings file y User settings file for data, una ruta se devuelve sólo si la opción de seguridad **Activar configuración de usuario en el archivo externo** se ha seleccionado en el diálogo "Propiedades de la base".

Ejemplo

Usted quiere obtener la ruta del último archivo de backup:

```
C_TEXT($path)
$path:=Get 4D file(Last backup file)
 // $path = "C:\Backups\Countries\Countries[0025].4BK" for example
```

Get 4D folder

Get 4D folder {(carpeta {; *})} -> Resultado

Parámetro	Tipo		Descripción
carpeta	Entero largo	➡	Tipo de carpeta (si se omite = carpeta activa 4D)
*	Operador	➡	Devuelve la carpeta de la base local
Resultado	Cadena	↩	Ruta de acceso a la carpeta designada

Descripción

El comando **Get 4D folder** devuelve la ruta de acceso a la carpeta 4D activa de la aplicación actual o de la carpeta del entorno 4D especificada por el parámetro *carpeta*, si se pasa. Este comando le permite obtener la ruta de acceso actual a las carpetas utilizadas por la aplicación 4D. Utilizando este comando, asegura que su código funcionará correctamente en todas las plataformas que se ejecuten en un sistema localizado.

En *carpeta*, puede pasar una de las siguientes constantes, ubicadas en el tema "**Entorno 4D**":

Constante	Tipo	Valor
4D Client database folder	Entero largo	3
Active 4D Folder	Entero largo	0
Current resources folder	Entero largo	6
Data folder	Entero largo	9
Database folder	Entero largo	4
Database folder Unix syntax	Entero largo	5
HTML Root folder	Entero largo	8
Licenses folder	Entero largo	1
Logs folder	Entero largo	7

Abajo encontrará una descripción de cada carpeta:

Notas preliminares sobre los nombres de las carpetas:

- {Disk} es el disco donde está instalado el sistema.
- La palabra Usuario representa el nombre del usuario que abrió la sesión.

Carpeta 4D activa

Las aplicaciones del entorno 4D utilizan una carpeta especifica para almacenar la siguiente información:

- Archivos de preferencias utilizados por las aplicaciones 4D
- Archivo Shortcuts.xml (atajos de teclado personalizados)
- Carpeta Macros v2 (macro comandos del editor de métodos)
- Carpeta Favorites v1x, por ejemplo Favorites v13 (rutas de acceso de las bases locales y remotas que se hayan abierto)

Con las aplicaciones 4D principales (4D y 4D Server), la carpeta 4D activa se llama **4D** y se encuentra por defecto en la siguiente ubicación:

- En Windows 7 y siguientes: *{Disco}:\Users\<nomUsuario>\AppData\Roaming\<nombreBase>D*
- Bajo Mac OS: *{Disco}:Usuarios:<nomUsuario>:Library:Application Support:<nombreBase*

Carpeta Licencias

Carpeta que contiene los archivos de licencia del equipo.

La carpeta **Licenses** se encuentra en la siguiente ubicación:

- En Windows 7 y superiores: *{Disco}:\Datos de programa\4D\Licences*
- En OS X: *{Disco}:Library:Application Support:4D:Licenses*

Notas:

- En el caso de una aplicación fusionada con 4D Volume Desktop, la carpeta de licencias está incluída en el paquete de la aplicación.
- Si la carpeta de licencias no puede crearse en el sistema por falta de autorización, se crea en la siguiente ubicación:
  - En Windows 7 y superiores: *{Disco}:\Usuarios\<userName>\AppData\Roaming\4D\Licenses*
  - En OS X: *{Disco}:Usuarios:<userName>:Library:Application Support:4D:Licenses*

Carpeta de datos

Ruta de la carpeta que contiene el archivo de datos actual. La ruta de acceso se expresa utilizando la sintaxis estándar de la plataforma actual.

Carpeta base 4D Client (máquinas clientes)

Carpeta de la base 4D creada en local en cada equipo 4D Client para guardar archivos y carpetas relacionadas con la base (recursos, plug-ins, carpeta Extras, etc.).

La carpeta **4D Client Database** está en la siguiente ubicación en cada equipo cliente:

- En Windows 7 y superiores: *{Disco}:\Usuarios\<nombreUsuario>\AppData\Local\4D\<NombreDeLaBase_Direccion>*
- En OS X: *{Disco}:Usuarios:<nombreUsuario>:Library:Caches:4D:<NombreDeLaBase_Direccion>*

Carpeta base (Database Folder)

Carpeta que contiene el archivo de estructura de la base. La ruta de acceso se expresa utilizando la sintaxis estándar de la plataforma actual.

Con la aplicación 4D Client, esta constante es estrictamente equivalente a la constante anterior *4D Client Database Folder*: el comando devuelve la ruta de acceso de la carpeta creada localmente.

Carpeta base sintaxis Unix (Database Folder Unix Syntax)

Carpeta que contiene el archivo de estructura de la base. Esta constante designa la misma carpeta que la anterior pero la ruta de acceso que devuelve se expresa utilizando sintaxis Unix (Posix), de tipo/Usuarios/... Esta sintaxis se usa principalmente cuando utiliza el comando **LAUNCH EXTERNAL PROCESS** bajo OS X.

Carpeta de Resources

Carpeta Resources de la base. Esta carpeta contiene los elementos adicionales (imágenes, textos) utilizados por la interfaz de la base. Un componente puede tener su propia carpeta Resources. La carpeta Resources está ubicada junto al archivo de estructura de la base.

En modo cliente/servidor, esta carpeta permite organizar la transferencia de datos personalizados (imágenes, archivos, subcarpetas...) entre el equipo servidor y los equipos clientes. El contenido de esta carpeta se actualiza automáticamente cada vez que un equipo cliente se conecta. Todos los mecanismos de referenciación asociados a la carpeta Resources son soportados en modo cliente/servidor (carpeta .lproj, XLIFF, imágenes...). Adicionalmente, 4D ofrece varias herramientas que pueden utilizarse para administrar y actualizar esta carpeta dinámicamente, particularmente un explorador de recursos.

Nota: si la carpeta Resources no existe para la base, se creará ejecutando el comando **Get 4D folder** con la constante Current Resources folder.

Carpeta Logs

La carpeta **Logs** de la base. Esta carpeta centraliza los archivos log de la base actual. Se crea al mismo nivel del archivo de estructura y contiene los siguientes archivos log:

- conversión de la base,
- peticiones al servidor web,
- verificación y reparación de los datos
- verificación y reparación de la estructura
- diario de actividades backup/restaurar
- depuración de los comandos
- peticiones 4D server (generadas en equipos cliente y en el servidor).

Nota: si la carpeta **Logs** no existe para la base, ejecutando el comando **Get 4D folder** con la constante Logs Folder.

Carpeta raíz HTML

Carpeta raíz HTML actual de la base. La ruta de acceso devuelta se expresa con la sintaxis estándar de la plataforma actual. La carpeta raíz HTML es la carpeta en la cual el servidor web de 4D busca las páginas y archivos web solicitados. Por defecto, se llama **WebFolder** y se encuentra junto al archivo de estructura. Su ubicación puede definirse en la página Web/Configuración de las Preferencias o dinámicamente vía el comando **WEB SET ROOT FOLDER**.
Si el comando **Get 4D folder** se llama desde un 4D remoto, la ruta devuelta es la del equipo remoto, no la de 4D Server.

El parámetro opcional * es útil en el caso de una arquitectura que utiliza componentes: permite determinar la base (local o componente) para la cual quiere obtener la ruta de acceso a la carpeta. Este parámetro sólo es válido para las carpetas Database Folder, Database Folder Unix Syntax y Current Resources folder. Se ignora en todos los demás casos.
Cuando el comando se llama desde un componente:

- Si pasa el parámetro *, el comando devuelve la ruta de acceso de la carpeta de la base local,
- Si no pasa el parámetro *, el comando devuelve la ruta de acceso de la carpeta Componentes.
  La carpeta de la base (Database Folder y Database Folder Unix Syntax) devuelta difiere en función del tipo de arquitectura del componente:
  - En el caso de una carpeta/paquete .4dbase, el comando devuelve la ruta de acceso de la carpeta/paquete .4dbase,
  - En el caso de un archivo .4db ó .4dc, el comando devuelve la ruta de acceso de la carpeta "Componentes",
  - En el caso de un alias o atajo, el comando devuelve la ruta de acceso de la carpeta que contiene la base matriz original. El resultado difiere de acuerdo al formato de esta base (carpeta/paquete .4dbase o archivo .4db/.4dc), como se describió anteriormente.

Cuando el comando se llama desde la base local, devuelve la ruta de acceso de la carpeta de la base local, sin importar si se pasa o no el parámetro *.

Ejemplo 1

Durante el inicio de una base monousuario, usted quiere cargar (o crear) sus propios parámetros y almacenarlos en un archivo ubicado en la carpeta 4D. Para hacer esto, en el método de base **On Startup Database Method**, puede escribir las siguientes líneas:

```
MAP FILE TYPES("PREF";"PRF";"Preferences file")
  ` Asocia el tipo de archivo PREF en Mac OS a la extensión de archivo .PRF en Windows
$vsPrefNombreDoc:=Get 4D folder+"MisPrefs"  ` Construir la ruta de acceso al archivo Preferencias
  ` Verificar si el archivo existe
If(Test path name($vsPrefNombreDoc+(".PRF"*Num(On Windows)))#Is a document)
   $vtPrefDocRef:=Create document($vsPrefNombreDoc;"PREF")  ` Si no, crearlo
Else
   $vtPrefDocRef:=Open document($vsPrefNombreDoc;"PREF")  ` Si sí, abrirlo
End if
If(OK=1)
  ` Procesar el contenido del documento
```

```
    CLOSE DOCUMENT($vtPrefDocRef)
Else
  ` Manejar el error
End if
```

Ejemplo 2

Este ejemplo ilustra el uso de la constante **Database Folder Unix Syntax** bajo Mac OS para listar el contenido de la carpeta de la base:

```
$posixpath:="\""+Get 4D folder(Database folder Unix syntax)+"\""
$myfolder:="ls -l "+$posixpath
$in:=""
$out:=""
$err:=""
LAUNCH EXTERNAL PROCESS($myfolder;$in;$out;$err)
```

Nota: bajo Mac OS, es necesario colocar las rutas de acceso entre comillas cuando contienen los nombres de archivos o de carpetas con espacios. La secuencia de escape "\" permite insertar el carácter comillas en la cadena. También puede utilizar la instrucción **Char(Double quote)**.

Variables y conjuntos del sistema

Si el parámetro *carpeta* no es válido o si la ruta de acceso devuelta está vacía, la varible sistema OK toma el valor 0.

Get database localization

Get database localization {(tipoLeng)} -> Resultado

Parámetro	Tipo		Descripción
tipoLeng	Entero largo	⇒	Tipo de lenguaje
Resultado	Cadena	⇐	Lenguaje actual de la base

Descripción

El comando **Get database localization** devuelve el lenguaje por defecto o el lenguaje de la base, especificado por *tipoLeng*, expresado en el estándar definido por la RFC 3066. Generalmente, el comando devuelve "en" para inglés, "es" para español, etc. Para mayor información sobre este estándar y los valores devueltos por este comando, por favor consulte el **Anexo C: Arquitectura XLIFF** en el manual de *Diseño*. p

Varios parámetros de idiomas diferentes pueden utilizarse simultáneamente en la aplicación. Para designar el parámetro a obtener, pase en *tipoLeng* una de las siguiente constantes, que se encuentran en el tema **Entorno 4D**:

Constante	Tipo	Valor	Comentario
Current localization	Entero largo	1	Lenguaje actual de la aplicación: lenguaje por defecto o lenguaje definido vía el comando **SET DATABASE LOCALIZATION**.
Default localization	Entero largo	0	Lenguaje definido automáticamente por 4D al inicio en función de la carpeta Resources y del entorno sistema (no modificable)
Internal 4D localization	Entero largo	3	Lenguaje utilizado por 4D para ordenaciones y comparaciones de textos (definido en las Preferencias de la aplicación).
User system localization	Entero largo	2	Lenguaje definido por el usuario actual del sistema.

Por defecto, si omite el parámetro *tipoLeng*, el comando devuelve el lenguaje por defecto (0).
El lenguaje actual de la base permite definir la carpeta .lproj en la que el programa va a buscar loe elementos localizados de la base de datos. 4D determina automáticamente el lenguaje actual al iniciarse la base de acuerdo a los contenidos de la carpeta **Recursos** y del entorno del sistema. El principio consiste en que 4D carga la primera carpeta .lproj de la base que corresponde al lenguaje de referencia, con el siguiente orden de prioridades:

1. Lenguaje del sistema (en Mac OS, varios idiomas pueden ser definidos con un orden de preferencia, 4D utiliza este parámetro).

2. Lenguaje de la aplicación 4D.

3. Inglés

4. Primer lenguaje encontrado en la carpeta **Recursos**.

Nota: si la base no contiene una carpeta .lproj, 4D aplica el siguiente orden de prioridad: 1. Lenguaje del sistema 2. Inglés (si el lenguaje del sistema no puede identificarse).

Get database measures

Get database measures {(opciones)} -> Resultado

Parámetro	Tipo		Descripción
opciones	Objeto	➡	Opciones de retorno
Resultado	Objeto	⮌	Objeto que contiene las medidas de la base

Descripción

El comando **Get database measures** le permite obtener información detallada acerca de los eventos del motor de base de datos 4D. La información reenviada incluye los accesos lectura/escritura a los datos desde/hacia el disco o la memoria caché, así como también la utilización de los índices de la base, las búsquedas y las ordenaciones.

Get database measures devuelve un único objeto que contiene todas las medidas relevantes. El parámetro *opciones* le permite configurar las opciones para la información devuelta.

Presentación del objeto devuelto

El objeto devuelto contiene una sola propiedad llamada "DB", que tiene la siguiente estructura básica:

```
{
    "DB": {
        "diskReadBytes": {…},
        "cacheReadBytes": {…},
        "cacheMissBytes": {…},
        "diskWriteBytes": {…},

        "diskReadCount": {…},
        "cacheReadCount": {…},
        "cacheMissCount": {…},
        "diskWriteCount": {…},

        "dataSegment1": {…},
        "indexSegment": {…},

        "tables": {…},
        "indexes": {…}

    }
}
```

Este objeto está compuesto de ocho propiedades que contienen las medidas básicas ("diskReadBytes", "cacheReadBytes", "cacheMissBytes", "diskWriteBytes", "diskReadCount", "cacheReadCount", "cacheMissCount", "diskWriteCount") y propiedades adicionales ("dataSegment1 "," indexSegment ", "tables", "index") que también pueden contener propiedades elementales, pero a un nivel diferente y con un alcance diferente (ver más adelante).

Nota: una propiedad sólo está presente en el interior del objeto si éste recibe contenido. Cuando una propiedad no tiene ningún contenido, no está incluida en el objeto. Por ejemplo, si la base se ha abierto en modo de sólo lectura y los índices no se han utilizado, el objeto devuelto no contendrá "diskWriteBytes", "diskWriteCount", "indexSegment" e "indexes".

Propiedades elementales

Las propiedades elementales se pueden encontrar en los diferentes niveles del objeto DB. La propiedades elementales devuelven la misma información pero con alcances diferentes. Esta es una descripción de las propiedades elementales:

Nombre	Información devuelta
diskReadBytes	Bytes leídos desde el disco
cacheReadBytes	Bytes leídos desde la caché
cacheMissBytes	Bytes faltantes de la caché
diskWriteBytes	Bytes escritos en el disco
diskReadCount	Acceso en lectura desde el disco
cacheReadCount	Acceso en lectura desde la caché
cacheMissCount	Acceso de lectura faltante en la caché
diskWriteCount	Acceso en escritura en el disco

Las ocho propiedades elementales tienen la misma estructura del objeto, por ejemplo:

```
"diskReadBytes": {      "value": 33486473620,      "history": [          // optional          {"value": 52564,"time": -1665},          {"value": 54202,"time": -1649},
            …      ] }
```

- "**value**" (número): la propiedad "value" contiene un número que representa o bien una cantidad de bytes o un conteo de accesos. Básicamente, este valor es la suma de los valores del objeto "history" (aunque el objeto "history" no esté presente).

"**history**" (array de objetos): el array de objetos "history" es una compilación de valores de eventos agrupados por segundo. La propiedad "history" sólo está presente si se solicita a través del parámetro *opciones* (ver a continuación). El array history tendrá un máximo de 200 elementos. Cada elemento del array es en sí mismo un objeto que contiene dos propiedades: "value" y "time".

- "value" (número): cantidad de bytes o accesos manipulados durante el período de tiempo designado en la propiedad "time" asociada.

"time" (número): número de segundos transcurridos desde que se ha llamado la función. En el ejemplo anterior ("time": -1649) significa 1649 segundos atrás (o más precisamente entre 1649 y 1650 segundos). Durante este período de un segundo, 54,202 bytes se han leído en el disco.
El array history no contiene valores secuenciales (-1650, -1651, -1652, etc.) El valor anterior es -1665, lo que significa que nada se leyó en el disco en el período de 15 segundos entre 1650 y 1665.
Nota: Por defecto, el array contendrá solamente información útil.
Dado que el tamaño máximo del array es 200, si la base de datos se utiliza intensivamente (algo se lee cada segundo en el disco), la longitud máxima de la h6istoria será de 200 segundos. Por otro lado, si casi no pasa nada, excepto, por ejemplo, una vez cada 3 minutos, la longitud de la historia será de 600 minutos (3*200).
Este ejemplo puede ser representado en el siguiente diagrama:

4D internal history		Requested history: 30	
time	value	time	value
-2	4629	0	0
-4	7788	-1	0
-6	3718	-2	4629
-8	8814	-3	0
-10	3925	-4	7788
-12	775	-5	0
-14	6807	-6	3718
-16	3265	-7	0
-18	8086	-8	8814
-20	2539	-9	0
		-10	3925
		-11	0
		-12	775
		-13	0
		-14	6807
		-15	0
		-16	3265
		-17	0
		-18	8086
		-19	0
		-20	2539
		-21	-1
		-22	-1
		-23	-1
		-24	-1
		-25	-1
		-26	-1
		-27	-1
		-28	-1
		-29	-1
		-30	-1

dataSegment1 y indexSegment

Las propiedades "dataSegment1" e "indexSegment" pueden contener hasta cuatro propiedades elementales (cuando están disponibles):

```
"dataSegment1": {
    "diskReadBytes": {…},
    "diskWriteBytes": {…},
    "diskReadCount": {…},
    "diskWriteCount": {…}
    },
"indexSegment": {
    "diskReadBytes": {…},
    "diskWriteBytes": {…},
    "diskReadCount": {…},
    "diskWriteCount": {…}
    }
```

Estas propiedades devuelven la misma información que las propiedades elementales, pero detallada para cada archivo de la base:

- "dataSegment1" representa el archivo de datos .4dd en el disco
- "indexSegment" representa el archivo de índice .4dx en el disco

Por ejemplo, se puede obtener el siguiente objeto:

{ "DB": { "diskReadBytes": { "value": 718260 }, "diskReadCount": { "value": 229 }, "dataSegment1": { "diskReadBytes": { "value": 679092 }, "diskReadCount": { "value": 212 } }, "indexSegment": { "diskReadBytes": { "value": 39168 }, "diskReadCount": { "value": 17 } } }

Los valores devueltos corresponden a las fórmulas siguientes:
diskReadBytes.value = dataSegment1.diskReadBytes.value + indexSegment.diskReadBytes.value
diskWriteBytes.value = dataSegment1.diskWriteBytes.value + indexSegment.diskWriteBytes.value
diskReadCount.value = dataSegment1.diskReadCount.value + indexSegment.diskReadCount.value
diskWriteCount.value = dataSegment1.diskWriteCount.value + indexSegment.diskWriteCount.value

tables

La propiedad "tables" contiene tantas propiedades como tablas que hayan sido accedidas, ya sea en modo de lectura o escritura desde la apertura de la base. El nombre de cada propiedad es el nombre de la tabla en cuestión. Por ejemplo:

```
"tables": {      "Employees": {…)      "Companies": {…)      }
```

Cada objeto "table" contiene hasta 12 propiedades:

- Las primeras ocho propiedades son las *propiedades elementales* (ver más arriba) con los valores relacionados a la tabla implicada.
- Las otras dos propiedades, "records" y "blobs", también tienen las mismas ocho propiedades elementales, pero limitadas a ciertos tipos de campos:
- ◦ La propiedad "records" se refiere A todos los campos de la tabla (cadenas, fechas, números, etc.) a excepción de texto, imágenes y BLOBs.
  - La propiedad "blobs" se refiere a los campos de tipo texto, imagen y BLOB de la tabla.
- Una o dos propiedades adicionales, "fields" y "queries", también pueden estar presentes en función de las búsquedas y las ordenaciones realizadas en la tabla concerniente:
- ◦ La propiedad "fields" contiene el mayor número de atributos "nombre de campo" (que también son sub-objetos) como el número de campos que se utilizan para las búsquedas u ordenaciones.
  Cada objeto nombre de campo contiene:
  - un objeto "queryCount" (con o sin historia, dependiendo del parámetro *opciones*) si una búsqueda se ha realizado utilizando este campo
  - y/o un objeto "sortCount" (con o sin historia, dependiendo del parámetro *opciones*) si una ordenación se ha realizado utilizando este campo.

  Este atributo no se basa en el uso de índices; Todos los tipos de búsquedas y de ordenaciones se tienen en cuenta.
  Ejemplo: desde el lanzamiento de la base, varias búsquedas y ordenaciones se han llevado a cabo utilizando los campos *CompID*, *Name* y *FirstName*. TEl objeto devuelto contiene el siguiente sub-objeto "fields" (*opciones* con ruta y sin historial):

```
{      "DB": {          "tables": {              "Employees": {                  "fields": {                      "CompID": {
"queryCount": {                                  "value": 3                                  }                  },                      "Name": {
                                "queryCount": {                                  "value": 1                      },                          "sortCount": {
                                    "value": 3                              }                  },                      "FirstName": {
"sortCount": {                                  "value": 2                                  }                  } (...)
```

  Nota: el atributo "fields" se crea únicamente si una búsqueda o una ordenación se ha efectuado en la tabla; de lo contrario este atributo no estará presente.
- "queries" es un array de objetos que ofrece una descripción de cada búsqueda realizada en la tabla. Cada elemento del array contendrá tres atributos:
  - "queryStatement" (cadena): cadena de búsqueda (que contienen los nombres de los campo, pero no los valores buscados). Por ejemplo: "(Companies.PK_ID != ?)"
  - "queryCount" (objeto):
    - "value" (número): número de veces que la cadena de búsqueda se ha ejecutado, sin importar los valores buscados.
    - "history" (array de objetos) (si se solicita en *opciones*): propiedades del historial estándar "value" y "time"
  - "duration" (objeto) (si "value" es >0)
    - "value" (número): número de milisegundos
    - "history" (array de objetos) (si se solicita en *opciones*): propiedades del historial estándar "value" y "time"

Ejemplo: desde el momento en que se lanza la base, una sola búsqueda se ha realizado en la tabla Employees (*opciones* son con ruta y con historial):

```
{              "DB": {                  "tables": {                  "Employees": {                      "queries": [                      {
"queryStatement": "(Employees.Name == ?)",                      "queryCount": {<p>                      "value": 1,
"history": [                                  {                                  "value": 1,                                  "time": -2022
                                }                              ]                          },                      "duration": {                          "value":
2,                          "history": [                              {                                  "value": 2,
"time": -2022                                  }                              ]                          }                  }, (...)
```

Nota: el atributo "queries" se crea cuando al menos una búsqueda se ha efectuado en la tabla.

indexes

Este es el objeto más complejo. Todas las tablas a las cuales se ha tenido acceso utilizando uno o varios de sus índices se almacenan como propiedades y en el interior de las propiedades, también se incluyen los nombres de los índices utilizados como propiedades. Los índices de palabras claves aparecen por separado y sus nombres están seguidos por "*(Keyword)*". Por último, cada objeto nombre de índice contiene los ocho propiedades elementales relacionadas con este índice así como también un máximo de cuatro sub-objetos en función del uso del índice de la base desde su lanzamiento (cada sub-objeto sólo existe si al menos una operación correspondiente se ha realizado en algún momento desde el lanzamiento de la base).

Ejemplo: desde el lanzamiento de la base, varios índices del campo [Employees] EmpLastName han sido solicitados. Además, 2 registros fueron creados y 16 se suprimieron en la tabla [Companies]. Esta tabla tiene un campo "name" que está indexado. La tabla también se ha consultado y ordenado utilizando este campo. El objeto resultante contendrá:

```
"indexes": {      "Employees": {          "EmpLastName": {                  "diskReadBytes": {…},                  "cacheReadBytes": {…},
"cacheMissBytes": {…},                  "diskWriteBytes": {…},                  "diskReadCount": {…},                  "cacheReadCount": {…},
    "cacheMissCount": {…},                  "diskWriteCount": {…}              }          "EmpLastName (Keyword)": {...},          "index3Name": {…},
"index4Name": {…},          …          }          "Companies": {              "Name":              (...)                  "queryCount": {                      "value": 41
        },                  "sortCount": {                      "value": 3                  },                  "insertKeyCount": {                      "value": 2                  },
                "deleteKeyCount": {                      "value": 16                  }      table3Name: {…} }
```

Parámetro options

El parámetro *opciones* le permite personalizar la información real que devuelve el comando. En *opciones*, se pasa un objeto que puede contener hasta tres propiedades: "withHistory", "historyLength" y "path".

Propiedad	Tipo	Descripción
"withHistory"	Booleano	"true" significa que el objeto "history" será devuelto por la función dentro del objeto devuelto; "false" significa que el objeto devuelto por la función no contendrá el objeto "history"
"historyLength"	número	Define el tamaño del array history devuelto en segundos(*).
"path"	cadena|array de cadenas	Ruta completa de la propiedad específica o array de rutas completas de las propiedades específicas que desea obtener. Si pasa una cadena, sólo el valor correspondiente se devuelve en el objeto "DB" (si la ruta es válida). Ejemplo "DB.tables.Employees.records.diskWriteBytes". Cuando se pasa un array de cadenas, todos los valores correspondientes son devueltos en el objeto "DB" (si las rutas son válidas). Ejemplo: ["DB.tables.Employee.records.diskWriteBytes", "DB.tables.Employee.records.diskReadCount","DB.dataSegment1.diskReadBytes"]

(*) Como se describió anteriormente, la historia no se almacena como una secuencia de segundos, sino sólo con valores relevantes. Si no ocurre nada durante un par de segundos o más, nada se almacena y una brecha aparecerá en el array de historial interno. "time" puede contener, por ejemplo, -2, -4, -5, -10, -15, -30 con valores de 200, 300, 250, 400, 500,150. Si la propiedad "historyLength" está definida en 600 (10 minutos), luego el array devuelto contendrá 0, -1, -2, -3 ... -599 para "time" y sólo los valores -2, -4, - 5, -10, -15, -30 se llenarán. Todos los demás valores se pondrán en 0 (cero) como valor. También como se describió anteriormente, el único límite del array interno es el tamaño (200), no el tiempo. Esto significa que si hay una baja actividad para una propiedad específica, el tiempo más antiguo puede ser muy grande (por ejemplo: -3600 para hace una hora). También puede contener menos de 200 valores si la base se acaba de iniciar. En estos casos, si el tiempo del historial interno es menor al solicitado o si todos los valores importantes que ya se han definido en el array devuelto, el valor devuelto será -1.
Ejemplo: la base sólo se inició hace 20 segundos y la historia de la petición es de 60 segundos. Los valores devueltos entre 0 y -20 se definen con valores o ceros y los otros se establecerán con -1. Cuando un valor "-1" se devuelve, esto significa que, o bien el tiempo de solicitud es demasiado antiguo o que el valor ya no está en el array de historia interna (es decir, se ha llegado al límite de los 200 elementos y se han eliminado los valores mayores).

Cliente/servidor y componentes

Este comando devuelve la información sobre el uso de la base de datos. Esto significa que le devuelve un objeto válido con valores relevantes sólo cuando se le llama:

- en modo local 4D (si se llama desde un componente, devuelve información acerca de la base de datos del host)
- en el servidor en modo cliente/servidor.

Si el comando se llama desde un 4D remoto, luego el objeto se deja vacío.
En este contexto, si necesita obtener información acerca de la base de datos en el servidor, la forma más sencilla de realizar esta acción es crear un método con la opción "ejecutar en el servidor" activada.
Este principio también funciona para un componente: si el componente se utiliza en un contexto local 4D, devolverá la información sobre la base local; en un contexto 4D remoto, devolverá la información sobre la base del servidor.

Ejemplo 1

Usted desea obtener el objeto "history" en el objeto devuelto:

```
C_OBJECT($param)
C_OBJECT($measures)
OB SET($param;"withHistory";True)
$measures:=Get database measures($param)
```

Ejemplo 2

Sólo queremos saber el número total de bytes leídos en la memoria caché ("cacheReadBytes"):

```
C_OBJECT($oStats)
C_OBJECT($oParams)
OB SET($oParams;"path";"DB.cacheReadBytes")
$oStats:=Get database measures($oParams)
```

El objeto devuelto contiene, por ejemplo:

```
{ "DB": { "cacheReadBytes": { "value": 9516637 } } }
```

Ejemplo 3

Queremos obtener las medidas de bytes de caché en los últimos dos minutos:

```
C_OBJECT($oParams)
C_OBJECT($measures)
OB SET($oParams;"path";"DB.cacheReadBytes")
OB SET($oParams;"withHistory";True)
OB SET($oParams;"historyLength";2*60)
$measures:=Get database measures($oParams)
```

Get database parameter

Get database parameter ({tabla ;} selector {; valor}) -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del parámetro o Tabla por defecto si se omite este parámetro
selector	Entero largo	➡	Código del parámetro de la base
valor	Cadena	⬅	Valor alfa del parámetro
Resultado	Real	↩	Valor actual del parámetro

Descripción

El comando **Get database parameter** permite obtener el valor actual de un parámetro de la base 4D. Cuando el valor del parámetro es una cadena de caracteres, se devuelve en el parámetro *valorAlfa*.

El parámetro *selector* designa el parámetro a obtener. 4D ofrece las siguientes constantes predefinidas, en el tema **Parámetros de la base**:

Constante	Tipo	Valor	Comentario
Minimum Web process	Entero largo	6	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 -> 32 767 **Descripción**: número mínimo de proceso web a mantener en modo no contextual con 4D en modo local y 4D Server. Por defecto, el valor es 0 (ver a continuación).
Maximum Web process	Entero largo	7	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 -> 32 767 **Descripción**: número máximo de procesos web a mantener en modo no contextual con 4D en modo local y 4D Server. Por defecto, el valor es 10. En modo no contextual, para que el servidor web sea reactivo, 4D demora los procesos web 5 segundos y los reutiliza para ejecutar las posibles futuras peticiones HTTP. En términos de rendimiento, este principio es más ventajoso que crear un nuevo proceso para cada petición. Una vez se reutiliza un proceso web, se retrasa nuevamente 5 segundos. Cuando se alcanza el número máximo de procesos web, el proceso web se aborta. Si no se ha atribuido ninguna petición a un proceso web durante 5 segundos, el proceso se aborta, excepto si el número mínimo de procesos web se ha alcanzado (en cuyo caso los procesos se retrasan nuevamente). Estos parámetros le permiten ajustar el funcionamiento de su servidor web en función del número de peticiones y de la memoria disponible, como también de otros parámetros.
_o_Web conversion mode	Entero largo	8	**** *Selector desactivado* ****
_o_Database cache size	Entero largo	9	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: - **Descripción**: *constante obsoleta* (Conservada por razones de compatibilidad únicamente). Ahora se recomienda utilizar el comando **Get cache size**.
_o_4D Local mode scheduler	Entero largo	10	**** *Este selector es obsoleto y no debe utilizarse* ****
_o_4D Server scheduler	Entero largo	11	**** *Este selector es obsoleto y no debe utilizarse* ***
_o_4D Remote mode scheduler	Entero largo	12	**** *Este selector es obsoleto y no debe utilizarse* ***
4D Server timeout	Entero largo	13	**Alcance**: aplicación 4D si *valor* positivo **Se conserva entre dos sesiones**: sí si *valor* positivo **Valores posibles**: 0 -> 32 767 **Descripción**: valor del tiempo de espera antes de desconexión (timeout) de 4D Server a los equipos clientes. Por defecto, este valor se define en la página "Cliente-Servidor/Configuración" de la caja de diálogo Preferencias en el equipo servidor. El timeout del servidor define el periodo máximo de no respuesta del cliente "autorizado", por ejemplo si realiza una operación de bloqueo. Al terminar esta periodo, 4D Server desconecta al cliente. El selector 4D Server Timeout le permite asignar en el parámetro *valor* un nuevo timeout, expresado en minutos. Esta funcionalidad es particularmente útil para aumentar el valor del timeout antes de la ejecución en el equipo cliente de una operación de larga duración, tal como la impresión de un gran número de páginas, la cual puede causar un timeout inesperado. Tiene dos opciones: • Si pasa un valor **positivo** en el parámetro *valor*, define un timeout global y permanente: el nuevo valor se aplica a todos los procesos y se almacena en las Preferencias de la aplicación 4D (equivalente a cambiar en el diálogo Preferencias). • Si pasa un valor **negativo** en el parámetro *valor*, define un timeout lobal y temporal: el nuevo valor se aplica únicamente a los procesos llamantes (los otros procesos conservan los valores por defecto) y se restaura al valor por defecto tan pronto como el servidor recibe una señal de actividad del cliente, por ejemplo, cuando la operación termina. Esta opción es muy útil para administrar operaciones largas iniciadas por plug-ins 4D. Para definir una conexión "Sin timeout", pase 0 en *valor*. Ver el ejemplo 1.
4D Remote mode timeout	Entero largo	14	**Alcance** (antigua capa de red únicamente): aplicación 4D si valor positivo **Se conserva entre dos sesiones**: sí si *valor* positivo **Descripción**: a utilizar en casos muy específicos. Valor del timeout otorgado por el equipo 4D remoto a la máquina 4D Server. Por defecto, este valor se define en la página "Cliente-Servidor/Configuración" de la caja de diálogo de Preferencias en el equipo remoto. El selector Timeout 4D mode distant no se tiene en cuenta si utiliza la antigua capa de red. Con la capa 4D *ServerNet* activada, se ignora: esta configuración es administrada por el selector Timeout 4D Server (13).

Constante	Tipo	Valor	Comentario
Port ID	Entero largo	15	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: no **Descripción**: Número de puerto TCP utilizado por el servidor web 4D con 4D en modo local y 4D Server. Por defecto, el valor es 80. El número de puerto TCP está definido en la página "Web/Configuración" de la caja de diálogo de las Propiedades de la base. Puede utilizar las constantes del tema para el parámetro *valor*. El selector Port ID se utiliza en el marco de servidores web 4D compilados y fusionados con 4D Desktop (sin acceso al modo Diseño). Para mayor información sobre el número de puerto TCP, consulte la sección **Parámetros del servidor web**
_o_IP Address to listen	Entero largo	16	**** *Selector inactivo, utilizar los comandos* ***WEB SET OPTION*** *y* ***WEB GET OPTION*** ****
Character set	Entero largo	17	**Alcance:** 4D local, 4D Server **Se conserva entre dos sesiones:** sí **Descripción:** Constante obsoleta (se conserva por compatibilidad únicamente). Ahora recomendamos utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Max concurrent Web processes	Entero largo	18	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores**: todo valor entre 10 y 32 000. El valor por defecto es 100. **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Client minimum Web process	Entero largo	19	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 6 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Client maximum Web process	Entero largo	20	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 7 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Client Max Web requests size	Entero largo	21	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 27 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Client port ID	Entero largo	22	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 15 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
_o_Client IP address to listen	Entero largo	23	**** *Selector inactivo, utilizar los comandos* ***WEB SET OPTION*** *y* ***WEB GET OPTION*** ****
Client character set	Entero largo	24	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 17 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir los valores sólo para algunos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Client max concurrent Web proc	Entero largo	25	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 18 **Descripción**: permite especificar esta parámetro para las máquinas 4D remotas utilizadas como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir este valor sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Maximum Web requests size	Entero largo	27	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores posibles**: 500 000 a 2 147 483 648. **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.

Constante	Tipo	Valor	Comentario
4D Server log recording	Entero largo	28	**Alcance**: 4D Server, 4D remoto **Se conserva entre dos sesiones**: no **Valores posibles**: 0 ó de 1 a X (0 = no grabar, 1 a X = número secuencial, añadido al nombre del archivo). **Descripción**: inicia o detiene la grabación de las peticiones estándar recibidas por 4D Server (excluyendo las peticiones web). Por defecto, el valor es 0 (no se graban las peticiones). 4D Server le permite grabar cada petición recibida por el equipo servidor en un archivo de historial. Cuando este mecanismo está activo, el archivo de historial se crea junto al archivo de estructura de la base. Su nombre es "4DRequestsLog_X," donde X es el número secuencial del historial. Una vez el archivo alcanza un tamaño de 10 MB, se cierra y se genera un nuevo archivo, con un número secuencial incrementado. Si existe un archivo con el mismo nombre, se reemplaza directamente. Puede definir el número de inicio de la secuencia utilizando el parámetro *valor*. Este archivo texto almacena en formato tabulado simple diferente información sobre cada petición: hora, número de proceso, usuario, tamaño de la petición, duración del proceso, etc. Esta información puede ser útil particularmente durante la fase de afinamiento de la aplicación o con fines estadísticos. Por ejemplo puede importarse, en un software de hoja de cálculo para procesarse.
_o_Web Log recording	Entero largo	29	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Client Web log recording	Entero largo	30	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 = No grabar (por defecto), 1 = Registrar en formato CLF, 2 = Registrar en formato DLF, 3 = Registrar en formato ELF, 4 = Registrar en formato WLF. **Descripción**: inicia o detiene la grabación de las peticiones web recibidas por los servidores web de todos los equipos cliente. Por defecto, el valor es 0 (no se graban las peticiones). El funcionamiento de este selector es idéntico al del selector 29; sin embargo, aplica a todos los equipos 4D remotos utilizados como servidores web. El archivo "logweb.txt", en este caso, automáticamente ubicado en la subcarpeta Logs de la base 4D remota (carpeta de caché). Si quiere definir los valores únicamente para ciertos equipos cliente, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Table sequence number	Entero largo	31	**Alcance**: *a*plicación 4D **Se conserva entre dos sesiones**: sí **Valores posibles**: todo valor de tipo entero largo. **Descripción**: este selector se utiliza para modificar o modificar u obtener el número único actual de los registros de la tabla pasada en parámetro. "Número actual" significa "último número utilizado": si modifica este valor utilizando **SET DATABASE PARAMETER**, el siguiente registro será el valor pasado + 1. Este nuevo número es el número devuelto por el comando **Sequence number** como también en todo campo de la tabla a la cual se asigna la propiedad "Autoincrementar" en el editor de estructura o vía SQL. Por defecto, este número único es definido por 4D y corresponde al orden de creación de los registros. Para información adicional, por favor consulte la documentación del comando **Sequence number**.
_o_Real display precision	Entero largo	32	**** *Selector desactivado* ****

Constante	Tipo	Valor	Comentario

Debug log recording	Entero largo	34	**Alcance**: Aplicación 4D
			Se conserva entre dos sesiones: No
			Descripción: inicia o detiene la grabación secuencial de los eventos a nivel de programación de 4D en el archivo 4DDebugLog, que se ubica automáticamente en la subcarpeta Logs de la base de datos, junto al archivo de estructura. Un nuevo formato texto tabulado, más compacto se utiliza en el archivo de registro de eventos "4DDebugLog [_n].txt" a partir de 4D v14 (donde _n es el número de segmento del archivo).
			Valores posibles: Entero largo contiene un campo de bits: valor = bit1(1)+bit2(2)+bit3(4)+bit4(8)+…).
			- Bit 1 (valor 1) permite activar el archivo (note que cualquier otro valor no nulo también lo activará)
			- Bit 2 (valor 2) permite solicitar los parámetros de llamada a los métodos y comandos.
			- Bit 3 (valor 4) permite activar el nuevo formato tabulado.
			- Bit 4 (valor 8) permite desactivar la escritura inmediata de cada operación en el disco (activado por defecto). La escritura inmediata es menor rápida y más eficaz por ejemplo para buscar las causas de un fallo. Si desactiva este modo, el contenido del archivo será más compacto y se generará más rápidamente.
			- Bit 5 (valor 16) desactiva el registro de llamadas de plug-ins (activado por defecto).
			En el formato no tabulado (anterior), los tiempos de ejecución se expresaban en milisegundos y el valor "< ms" se muestra si una operación se ejecuta en menos de un milisegundo.
			En el nuevo formato tabulado, los tiempos de ejecución se expresan en microsegundos.
			Ejemplos:
			SET DATABASE PARAMETER (34;1) // activa el archivo modo v13 sin los parámetros, con las duraciones
			SET DATABASE PARAMETER (34;2) // activa el archivo modo v13 con los parámetros y las duraciones
			SET DATABASE PARAMETER (34;2+4) // activa el archivo al formato v14 con los parámetros y las duraciones
			SET DATABASE PARAMETER (34;0) // desactiva el archivo
			Para evitar que el archivo registre demasiada información, puede restringir los comandos 4D a examinar con el selector 80, Log Command list.
			Esta opción se puede activar para todo tipo de aplicación 4D (4D todos los modos, 4D Server, 4D Volume Desktop), en modo interpretado o compilado.
			Nota: esta opción se ofrece únicamente con fines de depuración y no debe utilizarse en producción ya que puede afectar el rendimiento de la aplicación y saturar el disco duro. Para mayor información sobre este formato y el uso del archivo 4DDebugLog[_n].txt, por favor contacte al Soporte Técnico de 4D Inc.

Client Server port ID	Entero largo	35	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 a 65535 **Descripción**: número de puerto TCP donde el servidor 4D publica la base de datos (para conexión remota 4D). Por defecto, el valor es 19813. La personalización de este valor permite utilizar varias aplicaciones 4D cliente-servidor en la misma máquina con el protocolo TCP; en este caso, debe indicar un número de puerto diferente para cada aplicación. El valor se guarda en el archivo de estructura de la base. Puede definirse con 4D en modo local pero sólo se tiene en cuenta en configuración cliente servidor. Cuando modifica este valor, es necesario reiniciar el equipo servidor para que el nuevo valor sea tenido en cuenta.
Invert objects	Entero largo	37	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0, 1 ó 2 (0 = modo desactivado, 1 = modo automático, 2 = modo activo). **Descripción**: configuración del modo "inversión de los objetos" que permite invertir en modo Aplicación formularios, objetos, barras de menú, etc. cuando la base se muestra en Windows en un idioma de derecha a izquierda. Este modo también puede configurarse en la página Interfaz/Lenguajes de derecha a izquierda de las Propiedades de la base. • El valor 0 indica que el modo nunca ha sido activado, cualquiera que sea la configuración del sistema (corresponde al valor Nunca en las Propiedades de la base). • El valor 1 indica que el modo está activo o no en función de la configuración del sistema (corresponde al valor Automático en las Propiedades de la base). • El valor 2 indica que el modo está activo, cualquiera que sea la configuración del sistema (corresponde al valor Siempre en las Propiedades de la base). Para mayor información, consulte el manual de Diseño de 4D.
HTTPS Port ID	Entero largo	39	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.

Constante	Tipo	Valor	Comentario
Client HTTPS port ID	Entero largo	40	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 a 65535 **Descripción**: número de puerto TCP utilizado por los servidores web de los equipos clientes para conexiones seguras vía SSL (protocolo HTTPS). Por defecto, el valor es 443 (valor estándar). Este selector puede utilizarse para modificar por programación el puerto TCP utilizado por los servidores web de los equipos clientes para las conexiones seguras vía SSL (protocolo HTTPS). Por defecto, el valor es 443 (valor estándar). Este selector funciona exactamente igual que el selector 39; sin embargo, aplica a todos los equipos 4D remotos utilizados como servidores web. Si quiere modificar el valor de ciertos equipos clientes únicamente, utilice la caja de diálogo de Preferencias de 4D remoto.
Unicode mode	Entero largo	41	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 (modo compatibilidad) ó 1 (modo Unicode) **Descripción**: modo de ejecución actual de la base, relativo al conjunto de caracteres. 4D soporta el conjunto de caracteres Unicode pero puede funcionar en modo "compatibilidad" (basado en el conjunto de caracteres Mac ASCII). Por defecto, las bases de datos convertidas se ejecutan en modo compatibilidad (0) y las bases creadas a partir de la versión 11 o superior se ejecutan en modo Unicode. El modo de ejecución puede controlarse vía una opción de las Preferencias y también puede leerse o (con propósitos de realizar pruebas) modificarse vía este selector. La modificación de esta opción necesita que la base se reinicie para que sea tenida en cuenta. Note que dentro de un componente no es posible modificar este valor, sólo leerlo.
SQL Autocommit	Entero largo	43	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Posibles valores**: 0 (desactivación) o 1 (activación) **Descripción**: activación o desactivación del modo SQL auto-commit. Por defecto, el valor es 0 (modo desactivado) El modo auto-commit permite reforzar la integridad referencial de la base. Cuando este modo está activo, las peticiones *SELECT*, **INSERT**, **UPDATE** y *DELETE* (SIUD) se incluyen automáticamente en las transacciones cuando no se han ejecutado dentro de una transacción. Este modo igualmente puede definirse en las Preferencias de la base.
SQL Engine case sensitivity	Entero largo	44	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 (no se tienen en cuenta las mayúsculas y minúsculas) ó 1 (sensible a las mayúsculas y minúsculas) **Descripción**: activación o desactivación de la sensibilidad a mayúsculas y minúsculas para comparaciones de cadenas efectuadas por el motor SQL. Por defecto, el valor es 1 (sensible a las mayúsculas y minúsculas): el motor SQL diferencia entre mayúsculas y minúsculas y entre caracteres acentuados al comparar cadenas (ordenaciones y búsquedas). Por ejemplo "ABC"= "ABC" pero "ABC" # "Abc." En algunos casos, por ejemplo para alinear el funcionamiento del motor SQL con el del motor 4D, podría querer que las comparaciones de cadenas no tengan en cuenta las mayúsculas y minúsculas ("ABC"="Abc"). Esta opción también puede definirse en la **Página SQL** de las Preferencias de la base.
Client log recording	Entero largo	45	**Alcance**: equipo 4D remoto **Se conserva entre dos sesiones**: no **Valores posibles**: 0 ó de 1 a X (0 = no grabar, 1 a X = número secuencial, asociado al nombre del archivo). **Descripción**: inicia o detiene la grabación de peticiones estándar efectuadas por el equipo cliente 4D que ejecutó el comando (excluyendo las peticiones web). Por defecto, el valor es 0 (no se graban las peticiones). 4D le permite registrar el historial de peticiones realizadas por el equipo cliente. Cuando este mecanismo se activa, se crean dos archivos en el equipo cliente, en la subcarpeta Logs de la carpeta local de la base. Son llamados 4DRequestsLog_X y 4DRequestsLog_ProcessInfo_X, donde X es el número secuencial del historial. Una vez el archivo 4DRequestsLog alcanza un tamaño de 10 MB, se cierra y se genera uno nuevo, con un número secuencial incrementado. Si ya existe un archivo con el mismo nombre, se reemplaza directamente. Puede definir el número de inicio para la secuencia utilizando el parámetro *valor*. Estos archivos texto almacenan en formato tabulado simple diferente información relacionada con cada petición: hora, número de proceso, tamaño de la petición, duración del proceso, etc. Esta información es particularmente útil durante la fase de desarrollo de la aplicación o con fines estadísticos.

Constante	Tipo	Valor	Comentario
Query by formula on server	Entero largo	46	**Alcance**: tabla y procesos actuales **Se conserva entre dos sesiones**: no **Valores posibles**: 0 (utilizar la configuración de la base), 1 (ejecutar en cliente) o 2 (ejecutar en servidor) **Descripción**: ubicación de la ejecución de los comandos **QUERY BY FORMULA** y **QUERY SELECTION BY FORMULA** para la *tabla* pasada en parámetro. Cuando se utiliza una base en modo cliente-servidor, los comandos de búsqueda "por fórmula" pueden ejecutarse en el servidor o en el equipo cliente: • en bases creadas con 4D v11 SQL, estos comandos se ejecutan en el servidor. • en bases convertidas, estos comandos se ejecutan en el equipo cliente, como en las versiones anteriores de 4D. • en las bases convertidas, una preferencia específica permite modificar globalmente la ubicación de ejecución de estos comandos. Esta diferencia en ubicación de ejecución influye no sólo en el rendimiento de la aplicación (la ejecución en el servidor es generalmente más rápida) sino también en la programación. En efecto, el valor de los componentes de la fórmula (en particular las variables llamadas vía un método) varía de acuerdo al contexto de ejecución. Puede utilizar este selector para adaptar puntualmente el funcionamiento de su aplicación. Si pasa 0 en el parámetro *valor*, la ubicación de ejecución de los comandos de búsqueda "por fórmula" dependerá de la configuración de la base: en bases creadas con 4D v11 SQL, estos comandos se ejecutarán en el servidor. En bases convertidas, se ejecutarán en el equipo cliente o en el servidor en función de las preferencias de la base. Pase 1 ó 2 en *valor* para "forzar" la ejecución de estos comandos respectivamente en el equipo cliente o en el servidor. Consulte el ejemplo 2. **Nota:** si quiere activar las uniones "tipo SQL" (consulte el selector QUERY BY FORMULA Joins selector), siempre debe ejecutar las fórmulas en el servidor de manera que tengan acceso a los registros. Atención, en este contexto, la fórmula no debe contener llamadas a un método, de lo contrario pasará automáticamente al equipo remoto.
Order by formula on server	Entero largo	47	**Alcance**: tabla y procesos actuales **Se conserva entre dos sesiones**: no **Valores posibles**: 0 (utilizar la configuración de la base), 1 (ejecutar en el cliente) o 2 (ejecutar en el servidor) **Descripción**: ubicación de la ejecución del comando **ORDER BY FORMULA** para la tabla pasada en parámetro. Al utilizar una base en modo cliente-servidor, el comando **ORDER BY FORMULA** puede ejecutarse bien sea en el equipo servidor o en el cliente. Este selector puede utilizarse para especificar la ubicación de la ejecución de este comando (servidor o cliente). Este modo también puede definirse en las preferencias de la base. Para mayor información, consulte la descripción del selector 46, Query By Formula On Server. **Nota:** si quiere activar las uniones "tipo SQL" (consulte el selector QUERY BY FORMULA Joins selector), siempre debe ejecutar las fórmulas en el servidor de manera que tengan acceso a los registros. Atención, en este contexto, la fórmula no debe contener llamadas a un método, de lo contrario pasará automáticamente al equipo remoto.
Auto synchro resources folder	Entero largo	48	**Alcance**: equipo 4D remoto **Se conserva entre dos sesiones**: no **Valores posibles**: 0 (sin sincronización), 1 (auto sincronización) ó 2 (preguntar). **Descripción**: modo de sincronización dinámico de la carpeta *Resources* del equipo cliente 4D que ejecuta el comando con el servidor. Cuando el contenido de la carpeta *Resources* en el servidor se ha modificado o un usuario ha solicitado la sincronización (por ejemplo vía el explorador de recursos o siguiendo la ejecución del comando **NOTIFY RESOURCES FOLDER MODIFICATION**), el servidor notifica a los equipos cliente conectados. Tres modos de sincronización son posibles del lado del cliente. El selector Auto Synchro Resources Folder se utiliza para especificar el modo a utilizar por el equipo cliente para la sesión actual: • 0 (valor por defecto): sin sincronización dinámica (la petición de sincronización se ignora) • 1: sincronización dinámica automática • 2: visualización de una caja de diálogo en los equipos clientes, con la posibilidad de efectuar o rechazar la sincronización. El modo de sincronización también puede definirse globalmente en las Preferencias de la aplicación.

Constante	Tipo	Valor	Comentario
Query by formula joins	Entero largo	49	**Alcance**: Proceso actual **Se conserva entre dos sesiones**: no **Valores posibles**: 0 (utilizar configuración de la base), 1 (siempre utilizar relaciones automáticas) o 2 (utilizar las uniones SQL si es posible). **Descripción**: modo de funcionamiento de los comandos **QUERY BY FORMULA** y **QUERY SELECTION BY FORMULA** relativos al uso de "uniones SQL." En las bases de datos creadas a partir de la versión 11.2 de 4D v11 SQL, estos comandos efectúan uniones basados en el modelo de uniones SQL. Este mecanismo permite modificar la selección de una tabla en función de una búsqueda efectuada en otra tabla sin que las tablas estén conectadas por una relación automática (condición necesaria en las versiones anteriores de 4D). El selector QUERY BY FORMULA Joins permite definir el modo de funcionamiento de los comandos de búsqueda por fórmula para el proceso actual: • 0: Utilizar los parámetros actuales de la base (valor por defecto). En bases creadas a partir de la versión 11.2 de 4D v11 SQL, las "uniones SQL" siempre se activan para las búsquedas por fórmula. En bases de datos convertidas, este mecanismo no se activa por defecto por razones de compatibilidad pero puede implementarse vía una preferencia. • 1: Siempre utilizar relaciones automáticas (= funcionamiento de versiones anteriores de 4D). En este modo, una relación es necesaria para definir la selección de una tabla en función de búsquedas efectuadas en otra tabla. 4D no efectúa más "uniones SQL." • 2: Utilizar las uniones SQL si es posible (= funcionamiento o defecto de las bases creadas en versión 11.2 y superiores de 4D v11 SQL). En este modo, 4D establece "uniones SQL" para las búsquedas por fórmula cuando la fórmula se ajusta para ello (con dos excepciones, ver la descripción del comando **QUERY BY FORMULA** o **QUERY SELECTION BY FORMULA**). **Nota:** si quiere activar las uniones "tipo SQL" (consulte el selector QUERY BY FORMULA Joins selector), siempre debe ejecutar las fórmulas en el servidor de manera que tengan acceso a los registros. Atención, en este contexto, la fórmula no debe contener llamadas a un método, de lo contrario pasará automáticamente al equipo remoto.
HTTP compression level	Entero largo	50	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: no **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
HTTP compression threshold	Entero largo	51	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: no **Valores posibles**: todo valor de tipo entero largo **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Server base process stack size	Entero largo	53	**Alcance**: 4D Server **Se conserva entre dos sesiones**: no **Valores posibles**: entero largo positivo. **Descripción**: tamaño de la pila asignada a cada proceso del sistema preferente en el servidor, expresado en bytes. El tamaño por defecto es determinado por el sistema. Los procesos sistema preferente (procesos de tipo Proceso base 4D client) se cargan para controlar los procesos cliente 4D principales. El tamaño asignado por defecto a la pila de cada proceso preferente da facilidad de ejecución pero puede resultar consecuente cuando se crea un gran número de procesos (varios cientos). Por razones de optimización, este tamaño puede reducirse considerablemente si las operaciones efectuadas por la base lo permiten (por ejemplo si la base no efectúa ordenaciones de grandes cantidades de registros). Son posibles valores de 512 o incluso 256 KB. Sea cuidadoso, subdimensionar la pila es critico y puede afectar la operación de 4D Server. La definición de este parámetro debe hacerse con precaución y tener en cuenta las condiciones de uso de la base (número de registros, tipo de operaciones, etc.). Para que sea tenido en cuenta, este parámetro debe ejecutarse en el equipo servidor (por ejemplo en el **Método base On Server Startup**).

Constante	Tipo	Valor	Comentario
Idle connections timeout	Entero largo	54	**Alcance**: aplicación 4D a menos que valor sea negativo **Se conserva entre dos sesiones:** no **Valores posibles:** valor entero que expresa una duración en segundos. El valor puede ser positivo (nuevas conexiones) o negativo (conexiones existentes). Por defecto, el valor es 20. **Descripción**: máximo periodo de inactividad (timeout) para conexiones al motor de la base 4D y al motor SQL, así como también en modo *ServerNet* (nueva capa de red), al servidor de la aplicación 4D. Cuando una conexión inactiva alcanza este límite, se pone en espera automáticamente, lo cual congela la sesión cliente/servidor y cierra el socket de red. En la ventana de administración del servidor, el estado del proceso del usuario se indica como "Postponed". Este funcionamiento es totalmente transparente para el usuario: tan pronto como hay una nueva actividad en la conexión que está en espera, el socket se reabre automáticamente y la sesión cliente/servidor se restaura. Este parámetro permite, por una parte, economizar los recursos en el servidor: las conexiones en espera cierran el socket y liberan un proceso en el servidor. Por otra parte, esto le permite evitar pérdida de conexiones por el cierre de sockets por parte del firewall. Por esta razón, el valor del timeout para conexiones inactivas deber ser menor que el del firewall en este caso. Si pasa un valor positivo en *valor*, se aplicará a todas las nuevas conexiones en todos los procesos. Si pasa un valor negativo, se aplicará a las conexiones que se abran en el proceso actual. Si pasa 0, las conexiones inactivas no serán sometidas a un timeout. Este parámetro puede definirse del lado del servidor y del cliente. Si pasa dos duraciones diferentes, la más corta se tendrá en cuenta. Por lo general, no necesita cambiar este valor.
PHP interpreter IP address	Entero largo	55	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores**: cadena formateada del tipo "nnn.nnn.nnn.nnn" (por ejemplo "127.0.0.1"). **Descripción**: dirección IP utilizada localmente por 4D para comunicarse con el intérprete PHP vía FastCGI. Por defecto, el valor es "127.0.0.1". Esta dirección debe corresponder a la máquina donde en encuentra 4D. Este parámetro también puede definirse globalmente para todas las máquinas vía las Propiedades de la base. Para mayor información sobre el intérprete PHP, por favor consulte el manual de *Diseño*.
PHP interpreter port	Entero largo	56	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores**: valor de tipo entero largo positivo. Por defecto, el valor es 8002. **Descripción**: número de puerto TCP utilizado o por el intérprete PHP de 4D. Este parámetro también puede modificarse globalmente para todos los equipos vía las Propiedades de la base. Para mayor información sobre el intérprete PHP, consulte el manual de *Diseño*.
PHP number of children	Entero largo	57	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores**: valor de tipo entero largo positivo. Por defecto, el valor es 5. **Descripción**: número de procesos hijos a crear y mantener localmente por el intérprete PHP de 4D. Por razones de optimización, el intérprete PHP crea y utiliza un conjunto (pool) de procesos sistema llamados "procesos hijos" para procesar las peticiones de ejecución de scripts. Puede variar el número de procesos hijo de acuerdo a las necesidades de su aplicación. Este parámetro también puede modificarse globalmente para todos los equipos vía las Propiedades de la base. Para mayor información sobre el intérprete PHP, consulte el manual de *Diseño*. **Nota:** bajo Mac OS, todos los procesos hijos comparten el mismo puerto. Bajo Windows, cada proceso hijo utiliza un número de puerto específico. El primer número es el definido por el intérprete PHP; los otros procesos hijos lo incrementan. Por ejemplo, si el puerto por defecto es 8002 y usted lanza 5 procesos hijos, utilizarán los puertos 8002 a 8006.
PHP max requests	Entero largo	58	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores**: valor de tipo entero largo positivo. Por defecto, el valor es 500. **Descripción**: número máximo de peticiones aceptadas por el intérprete PHP. Cuando se alcanza este número máximo, el intérprete devuelve errores del tipo "servidor ocupado". Por razones de seguridad o rendimiento, puede modificar este valor. Este parámetro también puede modificarse globalmente para todos los equipos vía las Propiedades de la base. Para mayor información sobre este parámetro, consulte la documentación FastCGI-PHP. **Nota:** de parte de 4D, estos parámetros se aplican dinámicamente; no es necesario salir de 4D para que sean tenidos en cuenta. Por otra parte, si el intérprete PHP ya fue lanzado, será necesario salir y lanzarlo nuevamente, para que las modificaciones se tengan en cuenta.
PHP use external interpreter	Entero largo	60	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: no **Valores** : 0 = utilizar intérprete interno, 1 = utilizar intérprete externo **Descripción**: valor que indica si las peticiones PHP de 4D se envían al intérprete interno ofrecido por 4D o a un intérprete externo. Por defecto el valor es 0 (uso del intérprete ofrecido por 4D). Si quiere utilizar su propio intérprete PHP, por ejemplo para beneficiarse de módulos adicionales o de una configuración específica, pase 1 en *valor*. En este caso, 4D no lanza su intérprete interno en caso de peticiones PHP. El intérprete PHP personalizado debe haber sido compilado en FastCGI y estar ubicado en la misma máquina que el motor 4D. Note que en este caso, debe administrar completamente el intérprete; no será iniciado ni detenido por 4D. Este parámetro también puede modificarse globalmente para todas las máquinas vía las Propiedades de la base.

Constante	Tipo	Valor	Comentario
Maximum temporary memory size	Entero largo	61	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: entero largo positivo. **Descripción**: tamaño máximo de memoria temporal que 4D asignar a cada proceso, expresado en MB. Por defecto, el valor es 0 (sin tamaño máximo). 4D utiliza una memoria temporal especial dedicada a las operaciones de indexación y ordenación. Esta memoria conserva la memoria caché "estándar" durante operaciones masivas. Sólo se activa cuando es necesario. Por defecto, el tamaño de la memoria temporal está limitado únicamente por los recursos disponibles (en función de la configuración de memoria del sistema). Este mecanismo es conveniente para la mayoría de las aplicaciones. Sin embargo, en algunos contextos específicos, particularmente cuando una aplicación cliente-servidor efectúa simultáneamente un gran número de ordenaciones secuenciales, el tamaño de la memoria temporal puede aumentar críticamente, hasta volver el sistema inestable. En este contexto, fijar un tamaño máximo para la memoria temporal permite preservar el funcionamiento apropiado de la aplicación. En contraparte, la velocidad de ejecución podría afectarse: cuando se alcanza el tamaño máximo para un proceso, 4D utiliza archivos de discos, que pueden volver lentos los procesos. Para necesidades específicas tales como las descritas anteriormente, un tamaño máximo de 50 MB es generalmente un buen compromiso. Sin embargo, el valor ideal se determinará en función de las especificaciones de la aplicación y será generalmente el resultado de pruebas volumétricas en tiempo real.
SSL cipher list	Cadena	64	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: secuencia de cadenas separadas por dos puntos (por ejemplo "RC4-MD5:RC4-64-MD5:....") **Descripción:** lista de cifrado (*cipher list*) utilizada por 4D para el protocolo seguro. Esta lista modifica la prioridad de los algoritmos de cifrado implementados por 4D. Por ejemplo, puede pasar la siguiente cadena en el parámetro *valor*: "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". Para una descripción completa de la sintaxis para la lista cifrada, consulte la página de cifrado del sitio OpenSSL. Este parámetro es global para la aplicación (concierne al servidor HTTP, al servidor SQL, conexiones cliente/servidor, y también al cliente HTTP y a todos los comandos 4D que usan el protocolo seguro) pero es temporal (no se mantiene entre sesiones). Cuando la lista de cifrado se modifica, debe reiniciar el servidor correspondiente para que los nuevos parámetros sean tenidos en cuenta. Para reinicializar la lista de cifrado a su valor por defecto (guardado permanentemente en el archivo SLI), llame al comando **SET DATABASE PARAMETER** y pase una cadena vacía ("") en el parámetro *valor*. Por defecto, 4D utiliza el algoritmo de cifrado RC4. Si quiere utilizar el algoritmo AES (más reciente), pase la cadena siguiente en el parámetro valor: "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". **Nota:** con el comando **Get database parameter**, la lista de cifrado se devuelve en el parámetro opcional *valorAlfa* y el parámetro de retorno es siempre 0.
Cache unload minimum size	Entero largo	66	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: Entero largo positivo > 1. **Descripción**: tamaño mínimo de memoria a liberar del caché de la base de datos cuando el motor necesita hacer espacio para ubicar un objeto (valor en bytes). El propósito de este selector es reducir el número de liberaciones de datos de la caché con el fin de obtener un mejor rendimiento. Puede hacer variar este parámetro en función del tamaño de la caché y del de los bloques de datos manipulados en su base. Por defecto, si este selector no se utiliza, 4D descarga mínimo 10% de la caché en caso de que se necesite espacio.Alcance: Aplicación 4D Se conserva entre dos sesiones: No Valores posibles: Entero largo positivo > 1. Descripción: tamaño mínimo de memoria a liberar del caché de la base de datos cuando el motor necesita hacer espacio para ubicar un objeto (valor en bytes). El propósito de este selector es reducir el número de liberaciones de datos de la caché con el fin de obtener un mejor rendimiento. Puede hacer variar este parámetro en función del tamaño de la caché y del de los bloques de datos manipulados en su base. Por defecto, si este selector no se utiliza, 4D descarga mínimo 10% de la caché en caso de que se necesite espacio.
Direct2D status	Entero largo	69	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Descripción**: modo de activación de Direct2D bajo Windows. **Valores posibles:** una de las siguientes constantes (modo 3 por defecto): Direct2D Disabled (0): el modo Direct2D no está activo, la base funciona en el modo anterior (GDI/GDIPlus). Direct2D Hardware (1): uso de Direct2D en contexto de hardware gráfico en toda la aplicación 4D. Si este contexto no está disponible, uso del contexto de software gráfico Direct2D (excepto bajo Vista, en cuyo caso el modo GDI/GDIPlus se utiliza para un mejor rendimiento). Direct2D Software (3) (Modo por defecto): a partir de Windows 7, uso de Direct2D en contexto de software gráfico en toda la aplicación 4D. En Vista, por razones de rendimiento se utiliza el modo GDI/GDIPlus. **Nota de compatibilidad:** a partir de 4D v14, los modos híbridos se desactivan y redireccionan a los modos disponibles (el antiguo modo 2 es equivalente a 1; los antiguos modos 4 y 5 son equivalentes al modo 3).

Constante	Tipo	Valor	Comentario

Direct2D get active status	Entero largo	74	**Nota**: sólo puede utilizar este selector con el comando **Get database parameter** y su valor no puede definirse.
			Descripción: devuelve la implementación activa de Direct2D bajo Windows.
			Valores posibles: 0, 1, 2, 3, 4 o 5 (ver los valores del selector 69). El valor devuelto depende de la disponibilidad de Direct2D, del hardware y de la calidad Direct2D soportado por el sistema operativo.
			Por ejemplo, si ejecuta:
			`SET DATABASE PARAMETER(Direct2D status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D get active status)`
			- En Windows 7 y superiores, $mode vale 1 cuando el sistema detecta un hardware compatible con Direct2D; de lo contrario, $mode valdrá 3 (contexto software).
			- En Windows Vista, $mode valdrá 1 si el sistema detecta un hardware compatible con Direct2D; de lo contrario, $mode toma el valor 0 (desactivando Direct2D).
			- En Windows XP, $mode siempre valdrá 0 (no compatible con Direct2D).

Diagnostic log recording	Entero largo	79	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: 0 ó 1 (0 = no guardar,1 = guardar) **Descripción**: inicio o detención del registro del archivo de diagnóstico de 4D. Por defecto, el valor es 0 (no guarda). 4D permite guardar de manera continua en un archivo de diagnóstico un conjunto de eventos relativos al funcionamiento interno de la aplicación. La información contenida en este archivo está destinada a la actualización de las aplicaciones 4D y puede ser analizada con ayuda de los servicios técnicos de 4D. Cuando pasa 1 en este selector, el archivo de diagnóstico, llamado *NomBase.txt*, se crea automáticamente (o abre) en la carpeta **Logs** de la base. Una vez el archivo alcance un tamaño de 10 MB, se cierra y se genera un nuevo archivo *NomBase_N.txt*, con un número secuencial N incrementado. Note que es posible incluir la información personalizada en este archivo con ayuda del comando **LOG EVENT**.
Log command list	Cadena	80	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: cadena que contiene la lista de números de los comandos 4D a guardar (separados por dos puntos), "all" para guardar todos los comandos o "" (cadena vacía) para no guardar ninguno. **Descripción**: la lista de comandos 4D a guardar en el archivo de depuración (ver el selector 34, Debug Log Recording). Por defecto, se guardan todos los comandos 4D. Este selector permite guardar la cantidad de información almacenada en el archivo de depuración limitando los comandos 4D donde quiera guardar la ejecución.
Spellchecker	Entero largo	81	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: 0 (por defecto) = corrector macOS nativo (Hunspell desactivado), 1 = corrector Hunspell activo. **Descripción**: permite activar el corrector ortográfico Hunspell bajo macOS. Por defecto, en esta plataforma el corrector nativo está activo. Puede preferir utilizar el corrector Hunspell, por ejemplo, para unificar la interfaz de sus aplicaciones multiplataformas (bajo Windows, sólo el corrector Hunspell está disponible). Para mayor información, consulte **Corrección ortográfica**.
QuickTime support	Entero largo	82	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: Sí **Valores posibles**: 0 (por defecto) = QuickTime desactivado, 1 = QuickTime activado. **Descripción**:en 4D a partir de la v14, por defecto los codecs QuickTime ya no se soportan. Por compatibilidad, puede utilizar este selector para reactivarlos en su base. La modificación de esta opción requiere que la base se reinicie. Sin embargo, debe notar que en futuras versiones de 4D, se eliminará de forma permanente el soporte QuickTime.

Constante	Tipo	Valor	Comentario
Dates inside objects	Entero largo	85	**Alcance**: Proceso actual **Se conserva entre dos sesiones:** No **Valores posibles**: String type without time zone (0), String type with time zone (1), Date type (2) (por defecto) **Descripción**: define la forma en que se almacenan las fechas dentro de los objetos, así como también cómo se importan / exportan en JSON. Cuando el valor del selector es Date type (valor predeterminado para las bases creadas con 4D v17 y superior), las fechas 4D se almacenan con el tipo de fecha dentro de los objetos, con respecto a la configuración de fecha local. Cuando se convierte a formato JSON, los atributos de fecha se convertirán en cadenas que no incluyen un tiempo. (**Nota:** esta configuración se puede definir mediante la opción "Utilizar tipo de fecha en lugar del formato de fecha ISO en objetos" que se encuentra en **Página Compatibilidad** de la configuración de la base). Si pasa String type with time zone en este selector, convertirá las fechas 4D en cadenas ISO y tendrá en cuenta la zona horaria local. Por ejemplo, la conversión de la fecha 23/08/2013 le da "2013-08-22T22: 00: 000Z" en formato JSON cuando la operación se realiza en Francia durante el horario de verano (GMT+ 2). Este principio se ajusta al funcionamiento estándar de JavaScript. Esto puede ser una fuente de errores cuando desea enviar valores de fecha JSON a alguien en un huso horario diferente. Por ejemplo, cuando exporta una tabla usando **Selection to JSON** en Francia que se debe reimportar en los EE. UU. utilizando **JSON TO SELECTION**. Dado que las fechas se vuelven a interpretar en cada zona horaria, los valores almacenados en la base de datos serán diferentes. En este caso, puede modificar el modo de conversión de las fechas para que no tengan en cuenta la zona horaria pasando String type without time zone en este selector. La conversión de la fecha 23/08/2013 le dará "2013-08-23T00: 00: 00Z" en todos los casos.
Use legacy network layer	Entero largo	87	**Alcance:** 4D en modo local, 4D Server **Se conserva entre dos sesiones:** sí **Descripción:** fija u obtiene el estado actual de la capa de red antigua para las conexiones cliente/servidor. La capa de red antigua es obsoleta a partir de 4D v14 R5 y debe ser reemplazada progresivamente en sus aplicaciones por la capa de red *ServerNet*. *ServerNet* será requerida en próximas versiones 4D con el fin de beneficiarse de las futuras evoluciones de la red. Por razones de compatibilidad, la capa de red antigua aún se soporta para permitir una transición sin problemas para las aplicaciones existentes; (se usa por defecto en aplicaciones convertidas de una versión anterior a v14 R5). Pase 1 en este parámetro para utilizar la capa de red antigua (y desactivar *ServerNet*) para las conexiones cliente/servidor, y pase 0 para deshabilitar la red antigua (y utilizar *ServerNet*). Esta propiedad también se puede definir mediante la opción "Usar capa de red antigua " que se encuentran en **Página Compatibilidad** de las Propiedades de la base (ver **Opciones red y cliente-servidor**). En esta sección, también puede encontrar una discusión sobre la estrategia de migración. Le recomendamos que active *ServerNet* tan pronto como sea posible. Deberá reiniciar la aplicación para que este parámetro sea tenido en cuenta. No está disponible en 4D Server v14 R5 64-bit versión para OS X, que sólo soporta el *ServetNet*; (siempre devuelve 0). **Valores posibles:** 0 o 1 (0 = no utilizan capa de red antigua, 1 = uso capa de red antigua) **Valor por defecto:** 0 en bases de datos creadas con 4D v14 R5 o superior, 1 en bases de datos convertidas de 4D v14 R4 o anteriores.
SQL Server Port ID	Entero largo	88	**Alcance**: 4D modo local y 4D Server. **Se conserva entre dos sesiones**: Sí **Descripción**: permite leer o definir el número del puerto TCP utilizado por el servidor SQL integrado de 4D en modo local o 4D Server. Por defecto, el valor es 19812. Cuando se define este selector, la configuración de la base se actualiza. También puede definir el número del puerto TCP en la página "SQL" de la caja de diálogo de Propiedades de la base. **Valores posibles:** 0 a 65535. **Valor por defecto:** 19812
Circular log limitation	Entero largo	90	**Alcance**: 4D local, 4D Server. **Se conserva entre dos sesiones:** no **Valores posibles**: todo valor entero, 0 = conservar todos los registros **Descripción**: número máximo de archivos a conservar en rotación para cada tipo de registro. Por defecto, todos los archivos se conservan. Si pasa un valor *X*, solo los *X* archivos más recientes se conservan, el más antiguo se borra automáticamente cuando se crea uno nuevo. Este ajuste se aplica a cada uno de los siguientes archivos de registro: registros de peticiones (selectores 28 y 45), registro de depuración (selector 34), registro de eventos (selector 79), así como el historial de peticiones web y el historial de depuración Web (selectores 29 y 84 del comando **WEB SET OPTION**).
Number of formulas in cache	Entero largo	92	**Alcance**: aplicación 4D **Se conserva entre dos sesiones:** no **Valores posibles**: enteros largos positivos **Valor por defecto**: 0 (sin caché) **Descripción**: establece u obtiene el número máximo de fórmulas a conservar en la memoria caché de fórmulas, que es utilizado por el comando **EXECUTE FORMULA**. Este límite se aplica a todos los procesos, pero cada proceso tiene su propia caché de fórmulas. Ubicar las fórmulas en la caché acelera la ejecución del comando **EXECUTE FORMULA** en modo compilado, ya que cada fórmula en caché se tokeniza sólo una vez en este caso.Cuando se cambia el valor de la memoria caché, el contenido existente se restablecen incluso si el nuevo tamaño es más grande que el anterior. Una vez se alcanza el número máximo de fórmulas en la memoria caché, una nueva fórmula ejecutada borrará a la más antigua de la memoria caché (modo FIFO). Este parámetro sólo se tiene en cuenta en las bases o componentes compilados.

Constante	Tipo	Valor	Comentario
Cache flush periodicity	Entero largo	95	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones:** no **Valores posibles:** entero largo > 1 (segundos) **Descripción**: obtiene o establece la periodicidad del vaciado de la caché, expresado en segundos. La modificación de este valor prevalece sobre la opción **Vaciar caché cada X segundos** en **Página Base de datos/Memoria** de la configuración de la base para la sesión (que no se almacena en las Propiedades de la base).
Tips enabled	Entero largo	101	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: 0 = consejos desactivados, 1 = consejos activados (predeterminado) **Descripción**: define u obtiene el estado de visualización actual de los consejos para la aplicación 4D. De forma predeterminada, las sugerencias están activadas. Tenga en cuenta que este parámetro define todos los consejos 4D, es decir, los mensajes de ayuda de formulario y las sugerencias del editor de modo Diseño.
Tips delay	Entero largo	102	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: entero largo >= 0 (tics) **Descripción**: retraso antes de que se muestren las sugerencias una vez que el cursor del ratón se haya detenido en objetos con mensajes de ayuda adjuntos. El valor se expresa en tics (1/60 de segundo). El valor predeterminado es 45 tics (0.75 segundos).
Tips duration	Entero largo	103	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: entero largo >= 60 (tics) **Descripción**: duración máxima de visualización de una sugerencia. El valor se expresa en tics (1/60 de segundo). El valor predeterminado es 720 tics (12 segundos).
Min TLS version	Entero largo	105	**Alcance**: 4D Server, 4D Web Server y 4D SQL Server **Conservar entre dos sesiones**: No **Descripción**: se utiliza para especificar el nivel TLS (Transport Layer Security), que ofrece cifrado y autenticación de datos entre aplicaciones y servidores. Los valores definidos se aplican globalmente a la capa de red. Una vez modificado, el servidor debe reiniciarse para utilizar el nuevo valor. El protocolo mínimo predeterminado es TLSv1_2. Se rechazarán los intentos de conexión de clientes con un valor inferior al mínimo. **Valores posibles**: • TLSv1_0 - TLS 1.0, introducido en 1999 como una actualización de SSL v3.0. TLS 1.0 y SSL 3.0 no interoperan. • TLSv1_1 - TLS 1.1, introducido en 2006 como una actualización de TLS 1.0. Las mejoras incluyen mejor seguridad y manejo de errores, etc. • TLSv1_2 - TLS 1.2, introducido en 2008 como una actualización de TLS 1.1. Las mejoras incluyen una mayor flexibilidad, soporte adicional para el cifrado autenticado, etc. **NOTAS**: - El plugin 4D Internet Commands utiliza una capa de red diferente, por lo que este selector no tendrá ningún impacto en su versión TLS. - Se ignorarán los intentos de aplicar TLS a la capa de red heredada.
Times inside objects	Entero largo	109	**Alcance**: 4D local, 4D Server (todos los procesos) **Se conserva entre dos sesiones**: No **Valores posibles**: Times in seconds (0) (predeterminado), Times in milliseconds (1) **Descripción**: define la forma en que los valores de tipo hora se convierten y almacenan dentro de las propiedades de los objetos y los elementos de la colección, así como la forma en que se importan/exportan en JSON y en las áreas web. Por defecto, a partir de 4D v17, las horas se convierten y almacenan en número de segundos en los objetos. En versiones anteriores, los valores de tiempo se convertían y almacenaban como cantidad de milisegundos en esos contextos. Usar este selector puede ayudarlo a migrar sus aplicaciones volviendo a la configuración anterior si es necesario. **Nota**: los métodos ORDA y el motor SQL ignoran esta configuración, siempre suponen que los valores de tiempo son números de segundos.

Ejemplo

El siguiente método permite recuperar los valores actuales del planificador de 4D:

```
C_LONGINT($ticksbtwcalls;$maxticks;$minticks;$lparams)
If(Application type=4D Local Mode)  ` 4corriendo 4D en modo local
   $lparams:=Get database parameter(4D Local Mode Scheduler)
   $ticksbtwcalls:=$lparams  & 0x00ff
   $maxticks:=($lparams>>8) & 0x00ff
   $minticks:=($lparams>>16) & 0x00ff
End if
```

Get last update log path

Get last update log path -> Resultado

Parámetro	Tipo		Descripción
Resultado	Texto		Ruta de acceso del historial de actualización más reciente

Descripción

El comando **Get last update log path** devuelve la ruta de acceso completa del archivo de historial de actualización más reciente en la máquina donde se llama.

El historial de actualización es generado por 4D durante el proceso de actualización automático. Contiene información sobre los cambios realizados, así como los errores que se produjeron.

Este comando está destinado a ser utilizado en un proceso de actualización automática para una aplicación fusionada (servidor o monousuario). Para más información, consulte **Terminar y desplegar aplicaciones finales** en el manual de *Diseño*.

Get license info

Get license info -> Resultado

Parámetro	Tipo	Descripción
Resultado	Objeto	Información sobre la licencia activa

Descripción

El comando **Get license info** devuelve un objeto que ofrece información detallada sobre la licencia activa.
Si el comando se ejecuta en una aplicación 4D que no utiliza localmente una licencia (por ejemplo, 4D remoto), el comando devuelve un objeto Null.
El objeto devuelto contiene las siguientes propiedades:

```
{
    "name": string
    "licenseNumber": string
    "version": string
    "attributes": optional, array of strings
    "userName": string
    "userMail": string
    "companyName": string
    "platforms": array of strings
    "expirationDate": optional, object
   "renewalFailureCount": optional, number
    "products":[ //para cada producto de expansión registrado
        {
            "id": number
            "name": string
            "usedCount": number
            "allowedCount": number
            "rights": [
                {
                    "count": number
                    "expirationDate" optional, object
                }
            ]
        }
    ]
}
```

Nombre de la propiedad	Descripción	Ejemplos
name	Nombre comercial	"4D Developer Professional v16"
licenseNumber	Número de licencia	"4DDP16XXXXX1123456789"
version	Número de versión del producto	"16", "16R2"
attributes	Tipo(s) de licencia cuando sea aplicable (opcional)	["application","OEM"]
userName	Nombre de la cuenta de la tienda 4D	"John Smith"
userMail	Correo de la cuenta de la tienda 4D	"john.smith@gmail.com"
companyName	Nombre de la empresa de la cuenta de la tienda 4D	"Alpha Cie"
platforms	Plataforma(s) de licencia	["macOS", "windows"]
expirationDate	Fecha de vencimiento (opcional)	{"día":2, "mes":6, "año":2018}
renewalFailureCount	Número de intentos fallidos de renovación automática para al menos una de las licencias del producto (opcional)	3
products	Descripción de la licencia del producto (array de objetos, un elemento por licencia de producto)	
id	Número de licencia	Para conocer los valores disponibles, consulte el comando **Is license available**
name	Nombre de licencia	"4D Write - 4D Write Pro"
usedCount	Número de conexiones consumidas	8
allowedCount	Total de conexiones permitidas	15 (32767 significa ilimitado)
rights	Derechos para el producto (array de objetos, un elemento por fecha de vencimiento)	
count	Número de créditos permitidos	15 (32767 significa ilimitado)
expirationDate	Fecha de vencimiento (el mismo formato que el anterior)	{"día":1, "mes":11, "año":2017}

Ejemplo

Usted desea obtener información osobre su licencia 4D Server actual:

```
C_OBJECT($obj)
$obj:=Get license info
```

$obj puede contener, por ejemplo:

{ "name": "4D Server v16 R3", "licenseNumber": "xxxx", "version": "16R3", "userName": "John DOE", "userMail": "john.doe@alpha.com", "companyName": "Alpha", "platforms": ["macOS", "windows"], "expirationDate": {"day":1, "month":1, "year":2018}, "products":[{ "allowedCount": 15, "id": 808464697, "name": "4D Write - 4D Write Pro", "rights": [{ "count": 5, "expirationDate": {"day":1, "month":2, "year":2018} }, { "count": 10, "expirationDate": {"day":1, "month":11, "year":2017} }, { "count": 10, "expirationDate": {"day":1, "month":11, "year":2015} //vencido, no se cuenta }], "usedCount": 12 }, {...}] }

GET SERIAL INFORMATION

GET SERIAL INFORMATION (criterio ; usuario ; empresa ; conectados ; maxUsuarios)

Parámetro	Tipo		Descripción
criterio	Entero largo	⇐	Llave única del producto (encriptada)
usuario	Cadena	⇐	Nombre registrado
empresa	Cadena	⇐	Organización registrada
conectados	Entero largo	⇐	Número de usuarios conectados
maxUsuarios	Entero largo	⇐	Número máximo de usuarios conectados

Descripción

El comando **GET SERIAL INFORMATION** devuelve información sobre la serialización de la aplicación 4D.

- *llave*: identificación única del producto instalado. Un número único está asociado a una aplicación 4D (tal como 4D Server, 4D en modo local, 4D Desktop, etc.) instalada en un equipo. Desde luego, este número está encriptado.
- *usuario*: nombre de usuario de la aplicación como se definió al momento de la instalación.
- *empresa*: nombre de la empresa u organización como se definió durante la instalación.
- *conectados*: número de usuarios conectados al momento de la ejecución del comando.
- *maxUsuarios*: número máximo de usuarios conectados simultáneamente.

Nota: los dos últimos parámetros siempre devuelven 1 para las versiones monousuario de 4D, excepto en versiones de demostración (devuelven 0).

GET SERIAL INFORMATION es parte del esquema general de protección de los componentes implementado en 4D. Los desarrolladores de componentes pueden asociar una copia de su producto a una aplicación 4D instalada, con el fin de evitar copias ilegales.

La serialización funciona de la siguiente manera: un usuario que desea adquirir un componente envía al desarrollador su llave única generada por el comando **GET SERIAL INFORMATION**. Esta operación puede realizarse por intermedio de un formulario de Orden incluido en una versión de demostración del componente. La llave generada es única y está asociada a una aplicación 4D específica.

El desarrollador del componente puede entonces generar su propio número de serial combinando la llave y un código. El componente entregado ofrecerá una función de verificación si la información devuelta por **GET SERIAL INFORMATION** corresponde a este número de serial. De lo contrario, el usuario no podrá utilizar el componente.

Nota: los desarrolladores de plug-ins también pueden utilizar este esquema de protección. Para mayor información, consulte la documentación de 4D Plugin API Reference.

Get table fragmentation

Get table fragmentation (laTabla) -> Resultado

Parámetro	Tipo		Descripción
laTabla	Tabla	➡	Tabla para la cual obtener la tasa de fragmentación
Resultado	Real	⮌	Porcentaje de fragmentación

Descripción

El comando **Get table fragmentation** devuelve el porcentaje de fragmentación lógica de los registros de la tabla designada pro el parámetro *laTabla*.

La tasa de fragmentación de los registros indica si los registros se almacenan de manera ordenada en el archivo de datos. Una fragmentación muy alta, puede ralentizar considerablemente las ordenaciones y las búsquedas secuenciales en una tabla. Un porcentaje de fragmentación de 0 corresponde a una fragmentación nula. Una tasa de más del 20%, puede ser útil para compactar el archivo de datos.

Ejemplo

Este método de mantenimiento permite solicitar la compactación del archivo de datos en caso de que haya una fragmentación considerable en al menos una tabla de la base:

```
ToBeCompacted:=False
For($i ;1;Get last table number)
   If(Is table number valid($i))
      If(Get table fragmentation(Table($i)->)>20)
         ToBeCompacted:=True
      End if
   End if
End for
If(ToBeCompacted)
  // Pone un marcador de solicitud de compactación
End if
```

Is compiled mode

Is compiled mode {(*)} -> Resultado

Parámetro	Tipo		Descripción
*	Operador	⇒	Devuelve la información de la base local
Resultado	Booleano	⇐	Compilado (True), Interpretado (False)

Descripción

Is compiled mode prueba si la base se está ejecutando en modo compilado (True) o interpretado (False).
El parámetro opcional * es útil en caso de una arquitectura que utilice componentes: puede ser utilizada para determinar la base (local o componente) de cual quiere conocer el modo de ejecución.

• Cuando el comando se llama desde un componente:
- si se pasa el parámetro *, el comando devuelve **True** o **False** dependiendo del modo de ejecución de la base host,
- si no se pasa el parámetro *, el comando devuelve **True** o **False** dependiendo del modo de ejecución del componente.

• Cuando el comando se llama desde un método de una base local, devuelve **True** o **False** dependiendo del modo de ejecución de la base local.

Ejemplo

En una de sus rutinas, usted incluyó el código de depuración de la base, útil únicamente cuando está en modo interpretado. Puede preceder este código con una prueba que llama a **Is compiled mode**:

```
` ...
If(Not(Is compiled mode))
` Incluya el código para depurar su base aquí
End if
` ...
```

Is data file locked

Is data file locked -> Resultado

Parámetro	Tipo	Descripción
Resultado	Booleano	True = archivo/segmento bloqueado False = archivo/segmento no bloqueado

Descripción

El comando **Is data file locked** devuelve True si el archivo de datos de la base abierta o al menos uno de sus segmentos está bloquedo, es decir protegido contra escritura.

Por ejemplo si se usa en el método de base de datos **On Startup**, este comando permite la prevención de cualquier riesgo de apertura accidental de un archivo de datos bloqueado.

Ejemplo

Este método evitará la apertura de la base si el archivo de datos está bloqueado:

```
If(Is data file locked)
   ALERT("El archivo de datos está bloqueado. Imposible abrir la base.")
   QUIT 4D
End if
```

Is license available

Is license available {(licencia)} -> Resultado

Parámetro	Tipo		Descripción
licencia	Entero largo	⇒	Plug-in al cual realizar una prueba de validez de la licencia
Resultado	Booleano	⇐	True si el plug-in está disponible, sino False

Descripción

El comando **Is license available** le permite conocer la disponibilidad de un plug-in. Es útil, por ejemplo, para mostrar u ocultar funciones que necesitan de la presencia de un plug-in.

El comando **Is license available** puede utilizarse de tres maneras diferentes:

- El parámetro *licencia* se omite: en este caso, el comando devuelve **False** si la aplicación 4D está en modo demostración.
- Pase en el parámetro *licencia* una de la constantes del tema "Is license available":

Constante	Tipo	Valor
4D Client SOAP license	Entero largo	808465465
4D Client Web license	Entero largo	808465209
4D for OCI license	Entero largo	808465208
4D Mobile license	Entero largo	808464439
4D Mobile Test license	Entero largo	808465719
4D ODBC Pro license	Entero largo	808464946
4D SOAP license	Entero largo	808465464
4D View license	Entero largo	808465207
4D Web license	Entero largo	808464945
4D Write license	Entero largo	808464697

En este caso, el comando devuelve **True** si el plug-in correspondiente tiene una licencia disponible. El comando tiene en cuenta las licencias efectuadas en modo Diseño o vía el comando **SET PLUGIN ACCESS**.

Is license available devuelve **False** si el plug-in está funcionando en modo demostración.

- Pase en el parámetro *licencia* el número de identificación del recurso "4BNX" del plug-in. En este caso, el comando se comporta como se indicó anteriormente.

NOTIFY RESOURCES FOLDER MODIFICATION

NOTIFY RESOURCES FOLDER MODIFICATION
Este comando no requiere parámetros

Descripción

El comando **NOTIFY RESOURCES FOLDER MODIFICATION** permite "forzar" el envío por 4D Server de una notificación indicando a todos los puestos 4D conectados que el contenido de la carpeta Resources de la base ha sido modificado, de manera que puedan sincronizar su carpeta Resources local.
Este comando puede utilizarse particularmente para administrar la sincronización de las carpetas **Resources** de los equipos remotos después de que esta carpeta haya sido modificada por un procedimiento almacenado en el servidor.

Para mayor información sobre la administración de la carpeta **Resources** en modo remoto, consulte la Guía de referencia de 4D Server.

Sólo la información de modificación es enviada por este comando. Las máquinaa remotas reaccionarán en función del parámetro actual. Las opciones son:

- no sincronización de la carpeta **Resources** local durante la sesión,
- sincronización automática de la carpeta **Resources** local durante la sesión,
- mostrar una alerta con el fin de que el usuario efectúe una sincronización si lo desea.

El parámetro actual puede estar definido:

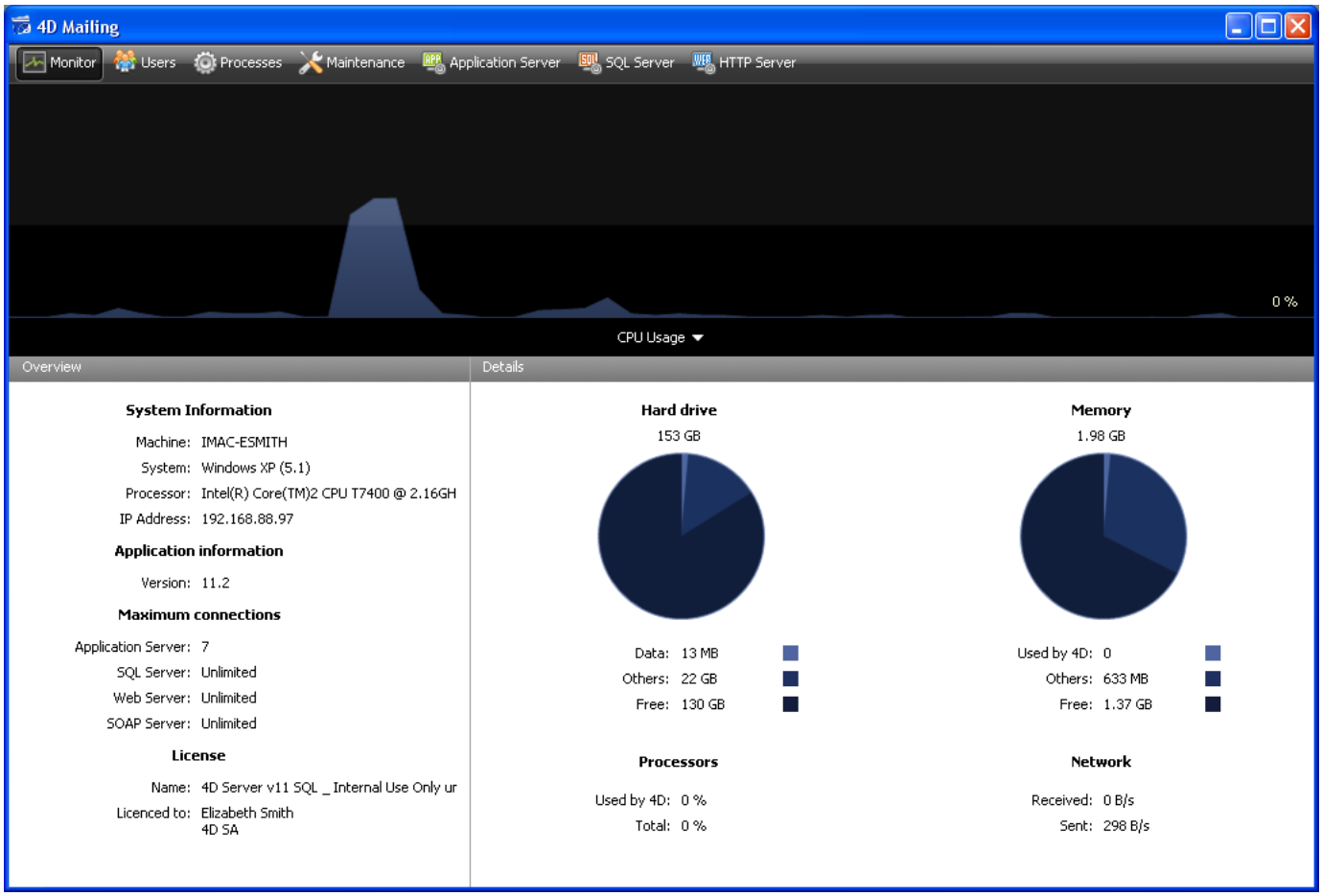
- a nivel global de la base vía el parámetro **Actualizar la carpeta "Resources" durante una sesión** de las Propiedades de la base. En este caso, se aplica a todos los equipos remotos;
- localmente, utilizando el comando **SET DATABASE PARAMETER** ejecutado en el equipo remoto (selector Auto Synchro Resources Folder). En este caso, "invalida" el de la base y se aplica únicamente al equipo remoto para la sesión.

OPEN ADMINISTRATION WINDOW

OPEN ADMINISTRATION WINDOW
Este comando no requiere parámetros

Descripción

El comando OPEN ADMINISTRATION WINDOW muestra la ventana de administración del servidor en el equipo que lo ejecuta. La ventana de administración de 4D Server permite visualizar los parámetros actuales y efectuar varias operaciones de mantenimiento (ver la Guía de referencia de 4D). A partir de la versión 11 de 4D Server, esta ventana puede mostrarse desde un equipo cliente:



Este comando debe llamarse en el contexto de una aplicación 4D conectada o de un 4D Server. No hace nada si:

- se llama en una aplicación 4D en modo local,
- es ejecutado por un usuario diferente al Diseñador o al Administrador (en este caso, se genera el error -9991, ver la sección **Errores de la base de datos**).

Ejemplo

Este es el código de un botón de administración:

```
If(Application type=4D local mode)
   OPEN SECURITY CENTER
   ` ...
End if
If(Application type=4D remote mode)
   OPEN ADMINISTRATION WINDOW
   ` ...
End if
If(Application type=4D Server)
   ` ...
   OPEN SECURITY CENTER
End if
```

Variables y conjuntos del sistema

Si el comando ha sido ejecutado correctamente, la variable sistema OK toma el valor 1. En caso contrario, toma el valor 0.

OPEN DATA FILE

OPEN DATA FILE (rutaAcceso)

Parámetro	Tipo		Descripción
rutaAcceso	Cadena	➡	Nombre o ruta de acceso completa del archivo de datos a abrir

Descripción

El comando **OPEN DATA FILE**permite cambiar el archivo de datos abierto por la aplicación 4D.

Pase el nombre o la ruta de acceso completa del archivo de datos a abrir en el parámetro *rutaAcceso* (archivo con el sufijo ".4DD"). Si pasa sólo el nombre del archivo, debe estar ubicado junto al archivo de estructura de la base.

Si la ruta de acceso establece un archivo de datos válido, 4D sale de la base y abre nuevamente con el archivo de datos especificado. En modo mono usuario **Semaphore** y **Método base On Startup** se llaman sucesivamente.

Advertencia: como este comando hace que la aplicación se cierre antes de abrir nuevamente con el archivo de datos especificado, debe utilizarse con precaución en el **Método base On Startup** o en un método llamado por este método base con el fin de no generar un bucle sin fin.

El comando se ejecuta de una manera asincrónica: después de su llamada, 4D continua ejecutando el resto del método. Luego, la aplicación se comporta como si el comando **Salir** hubiera sido seleccionado en el menú **Archivo** las cajas de diálogo abiertas son canceladas, los procesos abiertos tienen 10 segundos para terminar antes de que sean terminados, etc.

Antes de lanzar la operación, el comando verifica la validez del archivo de datos especificado. También, si el archivo ya estaba abierto, el comando verifica que corresponda a la estructura actual.

Si pasa una cadena vacía en *rutaAcceso*, el comando abrirá nuevamente la base de datos sin cambiar el archivo de datos.

4D Server: a partir de 4D v13, este comando puede utilizarse con 4D Server. En este contexto, hace una llamada interna a **QUIT 4D** en el servidor (lo que produce la aparición de una caja de diálogo en cada equipo remoto, indicando que el servidor está en proceso de salir) antes de abrir el archivo designado.

Ejemplo

En el contexto del despliegue de una aplicación fusionada, usted desea abrir o crear el archivo de datos usuario en el método base On Startup. En este ejemplo se utiliza el archivo de datos por defecto (ver **Gestión de archivo de datos en las aplicaciones finales**):

```
If(Data file="@default.4dd")
   If(Version type?? Merged application)
      If(Is data file locked)
         $dataPath:=Get 4D folder(Active 4D Folder)+"data.4dd"
 //Si un archivo de datos local ya existe
         If(Test path name($dataPath)=Is a document)
            OPEN DATA FILE($dataPath)  //abrirlo
         Else
            CREATE DATA FILE($dataPath)  //crearlo
         End if
      End if
   End if
End if
```

OPEN DATABASE

OPEN DATABASE (rutaArchivo)

Parámetro	Tipo		Descripción
rutaArchivo	Cadena	➡	Nombre o ruta de acceso completa del archivo de base de datos a abrir (.4db, .4dc, .4dbase o .4dlink)

Descripción

El comando **OPEN DATABASE** cierra la base de datos 4D actual y abre sobre la marcha de la base definida por *rutaArchivo*. Este comando es útil para realizar pruebas automáticas o para volver a abrir una base de forma automática después de una compilación.

En el parámetro *rutaArchivo*, pase el nombre o la ruta de acceso completa de la base a abrir. Puede utilizar los archivos con una de las siguientes extensiones:

- .4db (archivo de estructura interpretado),
- .4dc (archivo de estructura compilado),
- .4dbase (paquete OS X),
- .4dlink (archivo de acceso directo).

Si pasa únicamente el nombre de archivo, debe ser colocado en el mismo nivel que el archivo de estructura de la base actual.

Si la ruta de acceso establece una base de datos válida, 4D cierra la base en progreso y abre la base especificada. En el modo mono usuario, el **Semaphore** de la base cerrada y el **Método base On Startup** de la base abierta se llaman sucesivamente.

Atención: dado que este comando hace que la aplicación se cierre antes de la reapertura de la base especificada, no se recomienda su uso en el **Método base On Startup** o en un método llamado por este método base.

El comando se ejecuta de forma asíncrona: después de su llamada, 4D continúa ejecutando el resto del método.
Luego, la aplicación se comporta como si el comando **Salir** del menú **Archivo** estuviera seleccionado: las cajas de diálogo de apertura se cancelan, todos los procesos abiertos tienen 10 segundos para terminar antes de ser terminados, etc.

Si el archivo de la base objetivo no se encuentra o es inválido, se devuelve un error sistema estándar del administrador de archivos y 4D no hace nada.

Este comando se puede ejecutar desde una base de datos estándar únicamente. Si se llama desde una aplicación fusionada (monopuesto o servidor), se devuelve el error -10509 "No se puede abrir la base de datos".

Ejemplo

```
OPEN DATABASE("C:\\databases\\Invoices\\Invoices.4db")
```

OPEN SECURITY CENTER

OPEN SECURITY CENTER
Este comando no requiere parámetros

Descripción

El comando **OPEN SECURITY CENTER**muestra la ventana del Centro de seguridad y mantenimiento (CSM).

Dependiendo de los privilegios de acceso del usuario actual, ciertas funciones disponibles en esta ventana podrían desactivarse.

Nota: este comando funciona con el mismo principio que una llamada a **DIALOG** con el parámetro *: el CSM se muestra en una ventana y el comando devuelve inmediatamente el control al código 4D. Si el proceso actual termina, la ventana se cierra automáticamente mediante la simulación de un **CANCEL**. Por lo que debe gestionar su visualización a través del código del proceso en ejecución.

OPEN SETTINGS WINDOW

OPEN SETTINGS WINDOW (selector {; acceso}{; tipoParam})

Parámetro	Tipo		Descripción
selector	Cadena	➡	Llave que designa un tema o página o un grupo de parámetros de la caja de diálogo Preferencias
acceso	Booleano	➡	True=Bloquear las otras páginas de la caja de diálogo, False o si se omite=Dejar activas las otras páginas de la caja de diálogo
tipoParam	Entero largo	➡	0 o si se omite = Parámetros de estructura, 1 = Parámetros de usuario

Descripción

El comando **OPEN SETTINGS WINDOW** abre la caja de diálogo de Preferencias 4D o las Propiedades de la base actual y muestra los parámetros o la página correspondiente a la llave pasada en el parámetro *selector*.

El parámetro *selector* debe contener una llave indicando la caja de diálogo y la página a abrir. Esta llave se crea de esta forma: */Dialogo{/Pagina{/Parametros}}*. *Diálogo* indica la caja de diálogo a mostrar: puede pasar "4D" (para las Preferencias) o "Database" (Propiedades de la base). Por ejemplo, para indicar la página Compilador de las Propiedades de la base, *selector* debe contener "*/Database/Compiler*". La lista de llave que puede utilizar se ofrece a continuación. Si pasa una barra oblicua ("/") en *selector*, el comando muestra la primera página de la caja de diálogo de las Propiedades de la base.

El parámetro *acceso* le permite controlar las acciones del usuario en la caja de diálogo de Preferencias o de las Propiedades de la base bloqueando las otras páginas. Usted podría querer que el usuario pueda personalizar algunos parámetros pero evitar que otros puedan modificarse. En este caso, pasar True en el parámetro *acceso* significa que sólo la página especificada por el parámetro *selector* estará activa y modificable, mientras que el acceso a todas las otras páginas estará bloqueado (hacer clic en los botones de la barra de navegación no tendrá ningún efecto). Si pasa False u omite el parámetro *acceso*, todas las páginas de la caja de diálogo serán accesibles sin restricción.
El parámetro *tipoPropiedades* se tiene en cuenta en las bases configuradas en modo "Propiedades usuario" únicamente (en este modo, las "Propiedades usuario" personalizadas o "Propiedades usuario para el archivo de datos" se generan en un archivo externo y son utilizadas en lugar de las propiedades estándar, ver la sección **Configuración usuario** en el manual de *Diseño*). En este contexto, este parámetro le permite indicar si quiere acceder a la caja de diálogo de las "Propiedades de estructura", "Propiedades usuario", "Propiedades usuario para el archivo de datos". Puede pasar una de las siguientes constantes, del tema "**Entorno 4D**":

Constante	Tipo	Valor	Comentario
Structure settings	Entero largo	0	Acceso a las "propiedades estructura" (valor por defecto si el parámetro se omite). En este modo, los valores de *selector* utilizables son idénticos a los del modo estándar.
User settings	Entero largo	1	Acceso a las "propiedades usuario". En este modo, sólo ciertas llaves son utilizables en el parámetro *selector*.
User settings for data	Entero largo	2	Acceso a "Configuración usuario para archivo de datos, que es, configuración usuario almacenada en el mismo nivel que el archivo de datos. En este modo, sólo ciertas llaves se pueden utilizar con el parámetro *selector* (mismo subconjunto que la configuración usuario)

Si pasa una llave invalida, se muestra la primera página de la caja de diálogo de Propiedades de la base.

Llaves de rutas (modo estándar)

La siguiente es una lista de llaves que puede utilizarse en el parámetro *selector* en modo estándar, es decir con las "propiedades estructura":
/4D
/4D/General
/4D/Structure
/4D/Form editor
/4D/Method editor
/4D/Client-Server
/4D/Shortcuts
/Database
/Database/General
/Database/Mover
/Database/Interface
/Database/Interface/Developper
/Database/Interface/User
/Database/Interface/Shortcuts
/Database/Compiler
/Database/Database
/Database/Database/Data storage
/Database/Database/Memory and cpu
/Database/Database/International
/Database/Backup
/Database/Backup/Scheduler
/Database/Backup/Configuration
/Database/Backup/Backup and restore
/Database/Client-Server
/Database/Client-Server/Network
/Database/Client-Server/IP configuration
/Database/Web
/Database/Web/Config
/Database/Web/Options 1
/Database/Web/Options 2

/Database/Web/Log format
/Database/Web/Log scheduler
/Database/Web/Webservices
/Database/SQL
/Database/php
/Database/Compatibility
/Database/Security
Nota de compatibilidad: el comando continúa funcionando con las llaves definidas para las versiones 11; la correspondencia es establecida automáticamente por 4D. Sin embargo se recomienda reemplazar las llamadas antiguas por las llaves descritas anteriormente.

Llaves de rutas (modo Propiedades Usuario)

Estas son las llaves que se pueden utilizar en el parámetro *selector* en modo "Propiedades usuario":

/Database
/Database/Interface
/Database/Database/Memory and cpu
/Database/Client-Server
/Database/Client-Server/Network
/Database/Client-Server/IP configuration
/Database/Web
/Database/Web/Config
/Database/Web/Options 1
/Database/Web/Options 2
/Database/Web/Log format
/Database/Web/Log scheduler
/Database/Web/Webservices
/Database/SQL
/Database/php

Ejemplo 1

Apertura de la página "Métodos" de las Preferencias 4D":

```
OPEN SETTINGS WINDOW("/4D/Method editor")
```

Ejemplo 2

Acceso a los parámetros de los atajos de teclado en las Propiedades de la base con bloqueo de otras propiedades:

```
OPEN SETTINGS WINDOW("/Database/Interface/Shortcuts";True)
```

Ejemplo 3

Apertura de las Propiedades de la base en la primera página:

```
OPEN SETTINGS WINDOW("/")
```

Ejemplo 4

Acceso a la página interfaz de las Propiedades de la base en modo "Propiedades usuario":

```
OPEN SETTINGS WINDOW("/Database/Interface";1)
```

Variables y conjuntos del sistema

Si la caja de diálogo Preferencias/Propiedades se valida, la variable sistema OK devuelve 1; si se anula, OK devuelve 0.

PLUGIN LIST

PLUGIN LIST (arrayNumeros ; arrayNoms)

Parámetro	Tipo		Descripción
arrayNumeros	Array entero largo	⇐	Números de los plug-ins
arrayNoms	Array cadena	⇐	Nombres de los plug-ins

Descripción

El comando **PLUGIN LIST** llena los arrays *arrayNumeros* y *arrayNoms* con los números y los nombres de los plug-ins cargados por la aplicación 4D. Estos dos arrays son dimensionados y sincronizados automáticamente por el comando.

Nota: puede comparar los valores devueltos en el array *arrayNumeros* con las constantes del tema **Licencia disponible**.

PLUGIN LIST tiene en cuenta todos los plug-ins, incluyendo aquellos que están integrados (por ejemplo 4D Chart) y los plug-ins de terceras partes.

QUIT 4D

QUIT 4D {(tiempo)}

Parámetro	Tipo		Descripción
tiempo	Entero largo	➡	Tiempo en segundos antes de salir del servidor

Descripción

El comando QUIT 4D sale de la aplicación 4D y regresa al escritorio.
El proceso del comando es diferente si se ejecuta en 4D (modo local o remoto) o en 4D Server.

Con 4D modo local y 4D modo remoto

Después de llamar **QUIT 4D**, se detiene la ejecución del proceso actual, luego 4D efectúa las siguientes operaciones:

- Si hay un **Semaphore**, 4D comienza a ejecutar este método dentro de un nuevo proceso local. Por ejemplo, puede utilizar este método base para informar a otros procesos, vía comunicación interproceso, que deben cerrar (entrada de datos) o detener la ejecución de operaciones iniciadas por el **Método base On Startup** (conexión de 4D a otro servidor de bases de datos). Tenga en cuenta que 4D se cerrará en todo caso; el **Semaphore** puede realizar la limpieza y cierre de todas las operaciones, pero el cierre de la base es ineludible.
- Si no hay **Semaphore**, 4D cierra cada proceso en curso, uno por uno, sin distinción.

Si el usuario está introduciendo datos, los registros se cancelarán y no se guardarán.
Si quiere permitirle al usuario guardar las modificaciones efectuadas en las ventanas del proceso actual, puede utilizar la comunicación interproceso para indicar a todos los otros procesos de usuario que la base está a punto de cerrarse. Para hacer esto, puede adoptar dos estrategias:

- Efectuar estas operaciones desde el proceso actual antes de llamar **QUIT 4D.**
- Manejar estas operaciones desde el **Semaphore**.

También es posible una tercera estrategia. Antes de llamar **QUIT 4D**, pruebe si una ventana necesita validación; si ese es el caso, pida al usuario validar o cancelar esta ventana y luego seleccionar nuevamente Salir. Sin embargo, desde el punto de vista de la interfaz del usuario, las primeras dos estrategias son preferibles.
Nota: el parámetro *tiempo* no puede utilizarse con 4D en modo local o remoto.

Con 4D Server (procedimiento almacenado)

El comando **QUIT 4D** puede ejecutarse en el equipo servidor, en un proceso almacenado. En este caso, acepta el parámetro opcional *tiempo*.

El parámetro *tiempo* le da un tiempo de espera a 4D Server antes de que la aplicación se cierre realmente, permitiendo a los equipos cliente desconectarse. Debe pasar un valor en segundos en *tiempo*. Este parámetro sólo se tiene en cuenta durante una ejecución en el equipo servidor. Se ignora en 4D en modo local o remoto.
Si no pasa un parámetro para *tiempo*, 4D Server esperará hasta que todos los equipos cliente estén desconectados antes de salir.
A diferencia de 4D, el proceso de **QUIT 4D** por 4D Server es asincrónico: el método desde donde se llama el comando no se interrumpe después de su ejecución.
Si hay un **Método base On Server Shutdown**, se ejecuta después del tiempo definido por el parámetro *tiempo*, o después de que todos los clientes se hayan desconectado, dependiendo de sus parámetros.
La acción del comando **QUIT 4D** utilizada en un procedimiento almacenado es la misma del comando Salir del menú Archivo de 4D Server: provoca la aparición de una caja de diálogo en cada equipo cliente indicando que el servidor está a punto de cerrar.

Ejemplo

El método de proyecto siguiente está asociado al elemento de menú Salir en el menú Archivo.

```
` Método de proyecto M_SALIR

CONFIRM("¿Está seguro de que quiere salir?")
If(OK=1)
   QUIT 4D
End if
```

RESTART 4D

RESTART 4D {(demora {; mensaje})}

Parámetro	Tipo		Descripción
demora	Entero largo	➡	Tiempo de retardo (segundos) antes que 4D reinicie
mensaje	Cadena	➡	Texto a mostrar en los equipos clientes

Descripción

El comando **RESTART 4D** provoca el reinicio de la aplicación 4D actual.

Este comando es para uso en el contexto de una aplicación fusionada (cliente/servidor o mono puesto) y debe ser utilizado en junto con el comando **SET UPDATE FOLDER**. En este caso , el proceso de actualización automática se lanza: la nueva versión de la aplicación designada por **SET UPDATE FOLDER** reemplaza automáticamente la versión actual en el momento del reinicio resultantes de **RESTART 4D**. La ruta de acceso al archivo de datos se guarda y se utiliza de forma automática.

Si no se ha definido información de actualización utilizando el comando **SET UPDATE FOLDER** en la sesión actual, el comando reinicia simplemente la aplicación 4D con los archivos de estructura y de datos actuales.

Puede utilizar el parámetro *demora* para aplazar el reinicio de la aplicación con el fin de dar a los equipos cliente tiempo para desconectarse. Debe pasar un valor en segundos en *demora*. Si omite este parámetro, la aplicación servidor espera un máximo de 10 minutos, para que todas las aplicaciones cliente se desconecten. Después de este tiempo, todas las aplicaciones cliente se desconectan automáticamente.

Nota: los parámetros *demora* y *mensaje* sólo se tienen en cuenta con las aplicaciones servidor (se ignoran en el caso de una aplicación monopuesto o en un 4D remoto).

El parámetro opcional *mensaje* muestra un mensaje personalizado para las aplicaciones cliente conectadas.

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1, de lo contrario, toma el valor 0 y se reinicia la aplicación. Puede interceptar los errores generados por el comando utilizando un método instalado utilizando el comando **ON ERR CALL**.

SET DATABASE LOCALIZATION

SET DATABASE LOCALIZATION (codigoLeng {; *})

Parámetro	Tipo		Descripción
codigoLeng	Texto	➡	Selector del lenguaje
*	Operador	➡	Alcance del comando

Descripción

El comando **SET DATABASE LOCALIZATION** utilizado para modificar el lenguaje actual de la base para la sesión actual.

El lenguaje actual de la base de datos le permite especificar la carpeta .lproj donde el programa buscará los elementos localizados de la aplicación (textos e imágenes). Por defecto, 4D determina automáticamente el lenguaje actual según el contenido de la carpeta Resources y el entorno del sistema (ver la descripción del comando **Get database localization**). **SET DATABASE LOCALIZATION** permite modificar el idioma actual por defecto.

El comando no modifica el lenguaje de los formularios que ya están cargados, sólo los elementos mostrados después de la llamada al comando tendrán en cuenta la nueva configuración.

Pase el idioma que utilizará para la aplicación en *codigoLeng*, expresado en la norma definida por el RFC 3066, ISO639 y ISO3166. Normalmente, se debe pasar "fr" para francés, "es" para español, "en-us" para Inglés americano, y así sucesivamente. Para obtener más información sobre esta norma, por favor consulte el **Anexo C: Arquitectura XLIFF**.

Por defecto, el comando se aplica a todas las bases y componentes abiertos, para todos los procesos. El parámetro opcional *, si se pasa, especifica que el comando sólo se debe aplicar a la base de datos que lo llamó. Este parámetro se puede utilizar más concretamente, para especificar por separado el idioma de la base de datos y el de un componente.

Si el comando se ha ejecutado correctamente, la variable sistema OK toma el valor 1, de lo contrario, toma el valor 0.

Nota: de acuerdo con el RFC, el comando utiliza el "-" (guión) para separar el código del lenguaje y el código de la región, por ejemplo, "fr-ca" o "en-us". Por otra parte, las carpetas "lproj" de la carpeta **Resources** utilizan el "_" (guión bajo), por ejemplo "fr_ca.lproj" o "en_us.lproj".

4D Server: con 4D Server, los idiomas disponibles son los pertenecientes a la máquina remota que llamó al comando. Por lo tanto, debe asegurarse de que las carpetas **Resources** estén sincronizadas.

Ejemplo 1

Queremos definir el lenguaje de la interfaz como francés:

```
SET DATABASE LOCALIZATION("fr")
```

Ejemplo 2

La interfaz de su aplicación utiliza la cadena estática ":xliff:shopping". Los archivos XLIFF contienen particularmente la siguiente información:

- Carpeta FR:

```
<trans-unit id="15" resname="Shopping"> <source>Shopping</source> <target>Faire les courses</target> </trans-unit>
```

- Carpeta FR_CA folder:

```
<trans-unit id="15" resname="Shopping"> <source>Shopping</source> <target>Magasiner</target> </trans-unit>
```

```
SET DATABASE LOCALIZATION("fr")
 //La cadena ":xliff:shopping" muestra "Faire les courses"
SET DATABASE LOCALIZATION("fr-ca")
 //La cadena ":xliff:shopping" muestra "Magasiner"
```

SET DATABASE PARAMETER

SET DATABASE PARAMETER ({tabla ;} selector ; valor)

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla para la cual definir el parámetro o Tabla por defecto si se omite este parámetro
selector	Entero largo	⇒	Código del parámetro de la base a modificar
valor	Real, Cadena	⇒	Valor del parámetro

Descripción

El comando **SET DATABASE PARAMETER** permite modificar varios parámetros internos de la base de datos 4D.

El *selector* designa el parámetro a modificar. 4D ofrece constantes predefinidas, las cuales se ubican en el tema **Parámetros de la base**. La siguiente tabla lista cada constante e indica si si los cambios realizados se conservan entre dos sesiones:

Constante	Tipo	Valor	Comentario
Minimum Web process	Entero largo	6	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 -> 32 767 **Descripción**: número mínimo de proceso web a mantener en modo no contextual con 4D en modo local y 4D Server. Por defecto, el valor es 0 (ver a continuación).
Maximum Web process	Entero largo	7	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 -> 32 767 **Descripción**: número máximo de procesos web a mantener en modo no contextual con 4D en modo local y 4D Server. Por defecto, el valor es 10. En modo no contextual, para que el servidor web sea reactivo, 4D demora los procesos web 5 segundos y los reutiliza para ejecutar las posibles futuras peticiones HTTP. En términos de rendimiento, este principio es más ventajoso que crear un nuevo proceso para cada petición. Una vez se reutiliza un proceso web, se retrasa nuevamente 5 segundos. Cuando se alcanza el número máximo de procesos web, el proceso web se aborta. Si no se ha atribuido ninguna petición a un proceso web durante 5 segundos, el proceso se aborta, excepto si el número mínimo de procesos web se ha alcanzado (en cuyo caso los procesos se retrasan nuevamente). Estos parámetros le permiten ajustar el funcionamiento de su servidor web en función del número de peticiones y de la memoria disponible, como también de otros parámetros.
_o_Web conversion mode	Entero largo	8	**** *Selector desactivado* ****
_o_Database cache size	Entero largo	9	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: - **Descripción**: *constante obsoleta* (Conservada por razones de compatibilidad únicamente). Ahora se recomienda utilizar el comando **Get cache size**.
_o_4D Local mode scheduler	Entero largo	10	**** *Este selector es obsoleto y no debe utilizarse* ****
_o_4D Server scheduler	Entero largo	11	**** *Este selector es obsoleto y no debe utilizarse* ***
_o_4D Remote mode scheduler	Entero largo	12	**** *Este selector es obsoleto y no debe utilizarse* ***
4D Server timeout	Entero largo	13	**Alcance**: aplicación 4D si *valor* positivo **Se conserva entre dos sesiones**: sí si *valor* positivo **Valores posibles**: 0 -> 32 767 **Descripción**: valor del tiempo de espera antes de desconexión (timeout) de 4D Server a los equipos clientes. Por defecto, este valor se define en la página "Cliente-Servidor/Configuración" de la caja de diálogo Preferencias en el equipo servidor. El timeout del servidor define el periodo máximo de no respuesta del cliente "autorizado", por ejemplo si realiza una operación de bloqueo. Al terminar esta periodo, 4D Server desconecta al cliente. El selector 4D Server Timeout le permite asignar en el parámetro *valor* un nuevo timeout, expresado en minutos. Esta funcionalidad es particularmente útil para aumentar el valor del timeout antes de la ejecución en el equipo cliente de una operación de larga duración, tal como la impresión de un gran número de páginas, la cual puede causar un timeout inesperado. Tiene dos opciones: • Si pasa un valor **positivo** en el parámetro *valor*, define un timeout global y permanente: el nuevo valor se aplica a todos los procesos y se almacena en las Preferencias de la aplicación 4D (equivalente a cambiar en el diálogo Preferencias). • Si pasa un valor **negativo** en el parámetro *valor*, define un timeout lobal y temporal: el nuevo valor se aplica únicamente a los procesos llamantes (los otros procesos conservan los valores por defecto) y se restaura al valor por defecto tan pronto como el servidor recibe una señal de actividad del cliente, por ejemplo, cuando la operación termina. Esta opción es muy útil para administrar operaciones largas iniciadas por plug-ins 4D. Para definir una conexión "Sin timeout", pase 0 en *valor*. Ver el ejemplo 1.
4D Remote mode timeout	Entero largo	14	**Alcance** (antigua capa de red únicamente): aplicación 4D si valor positivo **Se conserva entre dos sesiones**: sí si *valor* positivo **Descripción**: a utilizar en casos muy específicos. Valor del timeout otorgado por el equipo 4D remoto a la máquina 4D Server. Por defecto, este valor se define en la página "Cliente-Servidor/Configuración" de la caja de diálogo de Preferencias en el equipo remoto. El selector Timeout 4D mode distant no se tiene en cuenta si utiliza la antigua capa de red. Con la capa 4D *ServerNet* activada, se ignora: esta configuración es administrada por el selector Timeout 4D Server (13).

Constante	Tipo	Valor	Comentario
Port ID	Entero largo	15	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: no **Descripción**: Número de puerto TCP utilizado por el servidor web 4D con 4D en modo local y 4D Server. Por defecto, el valor es 80. El número de puerto TCP está definido en la página "Web/Configuración" de la caja de diálogo de las Propiedades de la base. Puede utilizar las constantes del tema para el parámetro *valor*. El selector Port ID se utiliza en el marco de servidores web 4D compilados y fusionados con 4D Desktop (sin acceso al modo Diseño). Para mayor información sobre el número de puerto TCP, consulte la sección **Parámetros del servidor web**
_o_IP Address to listen	Entero largo	16	**** *Selector inactivo, utilizar los comandos* ***WEB SET OPTION*** *y* ***WEB GET OPTION*** ****
Character set	Entero largo	17	**Alcance:** 4D local, 4D Server **Se conserva entre dos sesiones:** sí **Descripción:** Constante obsoleta (se conserva por compatibilidad únicamente). Ahora recomendamos utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Max concurrent Web processes	Entero largo	18	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores**: todo valor entre 10 y 32 000. El valor por defecto es 100. **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Client minimum Web process	Entero largo	19	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 6 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Client maximum Web process	Entero largo	20	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 7 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Client Max Web requests size	Entero largo	21	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 27 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Client port ID	Entero largo	22	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 15 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
_o_Client IP address to listen	Entero largo	23	**** *Selector inactivo, utilizar los comandos* ***WEB SET OPTION*** *y* ***WEB GET OPTION*** ****
Client character set	Entero largo	24	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 17 **Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir los valores sólo para algunos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Client max concurrent Web proc	Entero largo	25	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: ver selector 18 **Descripción**: permite especificar esta parámetro para las máquinas 4D remotas utilizadas como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir este valor sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Maximum Web requests size	Entero largo	27	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores posibles**: 500 000 a 2 147 483 648. **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.

Constante	Tipo	Valor	Comentario
4D Server log recording	Entero largo	28	**Alcance**: 4D Server, 4D remoto **Se conserva entre dos sesiones**: no **Valores posibles**: 0 ó de 1 a X (0 = no grabar, 1 a X = número secuencial, añadido al nombre del archivo). **Descripción**: inicia o detiene la grabación de las peticiones estándar recibidas por 4D Server (excluyendo las peticiones web). Por defecto, el valor es 0 (no se graban las peticiones). 4D Server le permite grabar cada petición recibida por el equipo servidor en un archivo de historial. Cuando este mecanismo está activo, el archivo de historial se crea junto al archivo de estructura de la base. Su nombre es "4DRequestsLog_X," donde X es el número secuencial del historial. Una vez el archivo alcanza un tamaño de 10 MB, se cierra y se genera un nuevo archivo, con un número secuencial incrementado. Si existe un archivo con el mismo nombre, se reemplaza directamente. Puede definir el número de inicio de la secuencia utilizando el parámetro *valor*. Este archivo texto almacena en formato tabulado simple diferente información sobre cada petición: hora, número de proceso, usuario, tamaño de la petición, duración del proceso, etc. Esta información puede ser útil particularmente durante la fase de afinamiento de la aplicación o con fines estadísticos. Por ejemplo puede importarse, en un software de hoja de cálculo para procesarse.
_o_Web Log recording	Entero largo	29	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Client Web log recording	Entero largo	30	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 = No grabar (por defecto), 1 = Registrar en formato CLF, 2 = Registrar en formato DLF, 3 = Registrar en formato ELF, 4 = Registrar en formato WLF. **Descripción**: inicia o detiene la grabación de las peticiones web recibidas por los servidores web de todos los equipos cliente. Por defecto, el valor es 0 (no se graban las peticiones). El funcionamiento de este selector es idéntico al del selector 29; sin embargo, aplica a todos los equipos 4D remotos utilizados como servidores web. El archivo "logweb.txt", en este caso, automáticamente ubicado en la subcarpeta Logs de la base 4D remota (carpeta de caché). Si quiere definir los valores únicamente para ciertos equipos cliente, utilice la caja de diálogo de Preferencias de 4D en modo remoto.
Table sequence number	Entero largo	31	**Alcance**: *a*plicación 4D **Se conserva entre dos sesiones**: sí **Valores posibles**: todo valor de tipo entero largo. **Descripción**: este selector se utiliza para modificar o modificar u obtener el número único actual de los registros de la tabla pasada en parámetro. "Número actual" significa "último número utilizado": si modifica este valor utilizando **SET DATABASE PARAMETER**, el siguiente registro será el valor pasado + 1. Este nuevo número es el número devuelto por el comando **Sequence number** como también en todo campo de la tabla a la cual se asigna la propiedad "Autoincrementar" en el editor de estructura o vía SQL. Por defecto, este número único es definido por 4D y corresponde al orden de creación de los registros. Para información adicional, por favor consulte la documentación del comando **Sequence number**.
_o_Real display precision	Entero largo	32	**** *Selector desactivado* ****

Constante	Tipo	Valor	Comentario

Debug log recording	Entero largo	34	**Alcance**: Aplicación 4D
			Se conserva entre dos sesiones: No
			Descripción: inicia o detiene la grabación secuencial de los eventos a nivel de programación de 4D en el archivo 4DDebugLog, que se ubica automáticamente en la subcarpeta Logs de la base de datos, junto al archivo de estructura. Un nuevo formato texto tabulado, más compacto se utiliza en el archivo de registro de eventos "4DDebugLog [_n].txt" a partir de 4D v14 (donde _n es el número de segmento del archivo).
			Valores posibles: Entero largo contiene un campo de bits: valor = bit1(1)+bit2(2)+bit3(4)+bit4(8)+…).
			- Bit 1 (valor 1) permite activar el archivo (note que cualquier otro valor no nulo también lo activará)
			- Bit 2 (valor 2) permite solicitar los parámetros de llamada a los métodos y comandos.
			- Bit 3 (valor 4) permite activar el nuevo formato tabulado.
			- Bit 4 (valor 8) permite desactivar la escritura inmediata de cada operación en el disco (activado por defecto). La escritura inmediata es menor rápida y más eficaz por ejemplo para buscar las causas de un fallo. Si desactiva este modo, el contenido del archivo será más compacto y se generará más rápidamente.
			- Bit 5 (valor 16) desactiva el registro de llamadas de plug-ins (activado por defecto).
			En el formato no tabulado (anterior), los tiempos de ejecución se expresaban en milisegundos y el valor "< ms" se muestra si una operación se ejecuta en menos de un milisegundo.
			En el nuevo formato tabulado, los tiempos de ejecución se expresan en microsegundos.
			Ejemplos:
			SET DATABASE PARAMETER (34;1) // activa el archivo modo v13 sin los parámetros, con las duraciones
			SET DATABASE PARAMETER (34;2) // activa el archivo modo v13 con los parámetros y las duraciones
			SET DATABASE PARAMETER (34;2+4) // activa el archivo al formato v14 con los parámetros y las duraciones
			SET DATABASE PARAMETER (34;0) // desactiva el archivo
			Para evitar que el archivo registre demasiada información, puede restringir los comandos 4D a examinar con el selector 80, Log Command list.
			Esta opción se puede activar para todo tipo de aplicación 4D (4D todos los modos, 4D Server, 4D Volume Desktop), en modo interpretado o compilado.
			Nota: esta opción se ofrece únicamente con fines de depuración y no debe utilizarse en producción ya que puede afectar el rendimiento de la aplicación y saturar el disco duro. Para mayor información sobre este formato y el uso del archivo 4DDebugLog[_n].txt, por favor contacte al Soporte Técnico de 4D Inc.

Client Server port ID	Entero largo	35	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 a 65535 **Descripción**: número de puerto TCP donde el servidor 4D publica la base de datos (para conexión remota 4D). Por defecto, el valor es 19813. La personalización de este valor permite utilizar varias aplicaciones 4D cliente-servidor en la misma máquina con el protocolo TCP; en este caso, debe indicar un número de puerto diferente para cada aplicación. El valor se guarda en el archivo de estructura de la base. Puede definirse con 4D en modo local pero sólo se tiene en cuenta en configuración cliente servidor. Cuando modifica este valor, es necesario reiniciar el equipo servidor para que el nuevo valor sea tenido en cuenta.
Invert objects	Entero largo	37	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0, 1 ó 2 (0 = modo desactivado, 1 = modo automático, 2 = modo activo). **Descripción**: configuración del modo "inversión de los objetos" que permite invertir en modo Aplicación formularios, objetos, barras de menú, etc. cuando la base se muestra en Windows en un idioma de derecha a izquierda. Este modo también puede configurarse en la página Interfaz/Lenguajes de derecha a izquierda de las Propiedades de la base. • El valor 0 indica que el modo nunca ha sido activado, cualquiera que sea la configuración del sistema (corresponde al valor Nunca en las Propiedades de la base). • El valor 1 indica que el modo está activo o no en función de la configuración del sistema (corresponde al valor Automático en las Propiedades de la base). • El valor 2 indica que el modo está activo, cualquiera que sea la configuración del sistema (corresponde al valor Siempre en las Propiedades de la base). Para mayor información, consulte el manual de Diseño de 4D.
HTTPS Port ID	Entero largo	39	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.

Constante	Tipo	Valor	Comentario
Client HTTPS port ID	Entero largo	40	**Alcance**: todos los equipos 4D remotos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 a 65535 **Descripción**: número de puerto TCP utilizado por los servidores web de los equipos clientes para conexiones seguras vía SSL (protocolo HTTPS). Por defecto, el valor es 443 (valor estándar). Este selector puede utilizarse para modificar por programación el puerto TCP utilizado por los servidores web de los equipos clientes para las conexiones seguras vía SSL (protocolo HTTPS). Por defecto, el valor es 443 (valor estándar). Este selector funciona exactamente igual que el selector 39; sin embargo, aplica a todos los equipos 4D remotos utilizados como servidores web. Si quiere modificar el valor de ciertos equipos clientes únicamente, utilice la caja de diálogo de Preferencias de 4D remoto.
Unicode mode	Entero largo	41	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 (modo compatibilidad) ó 1 (modo Unicode) **Descripción**: modo de ejecución actual de la base, relativo al conjunto de caracteres. 4D soporta el conjunto de caracteres Unicode pero puede funcionar en modo "compatibilidad" (basado en el conjunto de caracteres Mac ASCII). Por defecto, las bases de datos convertidas se ejecutan en modo compatibilidad (0) y las bases creadas a partir de la versión 11 o superior se ejecutan en modo Unicode. El modo de ejecución puede controlarse vía una opción de las Preferencias y también puede leerse o (con propósitos de realizar pruebas) modificarse vía este selector. La modificación de esta opción necesita que la base se reinicie para que sea tenida en cuenta. Note que dentro de un componente no es posible modificar este valor, sólo leerlo.
SQL Autocommit	Entero largo	43	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Posibles valores**: 0 (desactivación) o 1 (activación) **Descripción**: activación o desactivación del modo SQL auto-commit. Por defecto, el valor es 0 (modo desactivado) El modo auto-commit permite reforzar la integridad referencial de la base. Cuando este modo está activo, las peticiones *SELECT*, **INSERT**, **UPDATE** y *DELETE* (SIUD) se incluyen automáticamente en las transacciones cuando no se han ejecutado dentro de una transacción. Este modo igualmente puede definirse en las Preferencias de la base.
SQL Engine case sensitivity	Entero largo	44	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 (no se tienen en cuenta las mayúsculas y minúsculas) ó 1 (sensible a las mayúsculas y minúsculas) **Descripción**: activación o desactivación de la sensibilidad a mayúsculas y minúsculas para comparaciones de cadenas efectuadas por el motor SQL. Por defecto, el valor es 1 (sensible a las mayúsculas y minúsculas): el motor SQL diferencia entre mayúsculas y minúsculas y entre caracteres acentuados al comparar cadenas (ordenaciones y búsquedas). Por ejemplo "ABC"= "ABC" pero "ABC" # "Abc." En algunos casos, por ejemplo para alinear el funcionamiento del motor SQL con el del motor 4D, podría querer que las comparaciones de cadenas no tengan en cuenta las mayúsculas y minúsculas ("ABC"="Abc"). Esta opción también puede definirse en la **Página SQL** de las Preferencias de la base.
Client log recording	Entero largo	45	**Alcance**: equipo 4D remoto **Se conserva entre dos sesiones**: no **Valores posibles**: 0 ó de 1 a X (0 = no grabar, 1 a X = número secuencial, asociado al nombre del archivo). **Descripción**: inicia o detiene la grabación de peticiones estándar efectuadas por el equipo cliente 4D que ejecutó el comando (excluyendo las peticiones web). Por defecto, el valor es 0 (no se graban las peticiones). 4D le permite registrar el historial de peticiones realizadas por el equipo cliente. Cuando este mecanismo se activa, se crean dos archivos en el equipo cliente, en la subcarpeta Logs de la carpeta local de la base. Son llamados 4DRequestsLog_X y 4DRequestsLog_ProcessInfo_X, donde X es el número secuencial del historial. Una vez el archivo 4DRequestsLog alcanza un tamaño de 10 MB, se cierra y se genera uno nuevo, con un número secuencial incrementado. Si ya existe un archivo con el mismo nombre, se reemplaza directamente. Puede definir el número de inicio para la secuencia utilizando el parámetro *valor*. Estos archivos texto almacenan en formato tabulado simple diferente información relacionada con cada petición: hora, número de proceso, tamaño de la petición, duración del proceso, etc. Esta información es particularmente útil durante la fase de desarrollo de la aplicación o con fines estadísticos.

Constante	Tipo	Valor	Comentario
Query by formula on server	Entero largo	46	**Alcance**: tabla y procesos actuales **Se conserva entre dos sesiones**: no **Valores posibles**: 0 (utilizar la configuración de la base), 1 (ejecutar en cliente) o 2 (ejecutar en servidor) **Descripción**: ubicación de la ejecución de los comandos **QUERY BY FORMULA** y **QUERY SELECTION BY FORMULA** para la *tabla* pasada en parámetro. Cuando se utiliza una base en modo cliente-servidor, los comandos de búsqueda "por fórmula" pueden ejecutarse en el servidor o en el equipo cliente: • en bases creadas con 4D v11 SQL, estos comandos se ejecutan en el servidor. • en bases convertidas, estos comandos se ejecutan en el equipo cliente, como en las versiones anteriores de 4D. • en las bases convertidas, una preferencia específica permite modificar globalmente la ubicación de ejecución de estos comandos. Esta diferencia en ubicación de ejecución influye no sólo en el rendimiento de la aplicación (la ejecución en el servidor es generalmente más rápida) sino también en la programación. En efecto, el valor de los componentes de la fórmula (en particular las variables llamadas vía un método) varía de acuerdo al contexto de ejecución. Puede utilizar este selector para adaptar puntualmente el funcionamiento de su aplicación. Si pasa 0 en el parámetro *valor*, la ubicación de ejecución de los comandos de búsqueda "por fórmula" dependerá de la configuración de la base: en bases creadas con 4D v11 SQL, estos comandos se ejecutarán en el servidor. En bases convertidas, se ejecutarán en el equipo cliente o en el servidor en función de las preferencias de la base. Pase 1 ó 2 en *valor* para "forzar" la ejecución de estos comandos respectivamente en el equipo cliente o en el servidor. Consulte el ejemplo 2. **Nota:** si quiere activar las uniones "tipo SQL" (consulte el selector QUERY BY FORMULA Joins selector), siempre debe ejecutar las fórmulas en el servidor de manera que tengan acceso a los registros. Atención, en este contexto, la fórmula no debe contener llamadas a un método, de lo contrario pasará automáticamente al equipo remoto.
Order by formula on server	Entero largo	47	**Alcance**: tabla y procesos actuales **Se conserva entre dos sesiones**: no **Valores posibles**: 0 (utilizar la configuración de la base), 1 (ejecutar en el cliente) o 2 (ejecutar en el servidor) **Descripción**: ubicación de la ejecución del comando **ORDER BY FORMULA** para la tabla pasada en parámetro. Al utilizar una base en modo cliente-servidor, el comando **ORDER BY FORMULA** puede ejecutarse bien sea en el equipo servidor o en el cliente. Este selector puede utilizarse para especificar la ubicación de la ejecución de este comando (servidor o cliente). Este modo también puede definirse en las preferencias de la base. Para mayor información, consulte la descripción del selector 46, Query By Formula On Server. **Nota:** si quiere activar las uniones "tipo SQL" (consulte el selector QUERY BY FORMULA Joins selector), siempre debe ejecutar las fórmulas en el servidor de manera que tengan acceso a los registros. Atención, en este contexto, la fórmula no debe contener llamadas a un método, de lo contrario pasará automáticamente al equipo remoto.
Auto synchro resources folder	Entero largo	48	**Alcance**: equipo 4D remoto **Se conserva entre dos sesiones**: no **Valores posibles**: 0 (sin sincronización), 1 (auto sincronización) ó 2 (preguntar). **Descripción**: modo de sincronización dinámico de la carpeta *Resources* del equipo cliente 4D que ejecuta el comando con el servidor. Cuando el contenido de la carpeta *Resources* en el servidor se ha modificado o un usuario ha solicitado la sincronización (por ejemplo vía el explorador de recursos o siguiendo la ejecución del comando **NOTIFY RESOURCES FOLDER MODIFICATION**), el servidor notifica a los equipos cliente conectados. Tres modos de sincronización son posibles del lado del cliente. El selector Auto Synchro Resources Folder se utiliza para especificar el modo a utilizar por el equipo cliente para la sesión actual: • 0 (valor por defecto): sin sincronización dinámica (la petición de sincronización se ignora) • 1: sincronización dinámica automática • 2: visualización de una caja de diálogo en los equipos clientes, con la posibilidad de efectuar o rechazar la sincronización. El modo de sincronización también puede definirse globalmente en las Preferencias de la aplicación.

Constante	Tipo	Valor	Comentario
Query by formula joins	Entero largo	49	**Alcance**: Proceso actual **Se conserva entre dos sesiones**: no **Valores posibles**: 0 (utilizar configuración de la base), 1 (siempre utilizar relaciones automáticas) o 2 (utilizar las uniones SQL si es posible). **Descripción**: modo de funcionamiento de los comandos **QUERY BY FORMULA** y **QUERY SELECTION BY FORMULA** relativos al uso de "uniones SQL." En las bases de datos creadas a partir de la versión 11.2 de 4D v11 SQL, estos comandos efectúan uniones basados en el modelo de uniones SQL. Este mecanismo permite modificar la selección de una tabla en función de una búsqueda efectuada en otra tabla sin que las tablas estén conectadas por una relación automática (condición necesaria en las versiones anteriores de 4D). El selector QUERY BY FORMULA Joins permite definir el modo de funcionamiento de los comandos de búsqueda por fórmula para el proceso actual: • 0: Utilizar los parámetros actuales de la base (valor por defecto). En bases creadas a partir de la versión 11.2 de 4D v11 SQL, las "uniones SQL" siempre se activan para las búsquedas por fórmula. En bases de datos convertidas, este mecanismo no se activa por defecto por razones de compatibilidad pero puede implementarse vía una preferencia. • 1: Siempre utilizar relaciones automáticas (= funcionamiento de versiones anteriores de 4D). En este modo, una relación es necesaria para definir la selección de una tabla en función de búsquedas efectuadas en otra tabla. 4D no efectúa más "uniones SQL." • 2: Utilizar las uniones SQL si es posible (= funcionamiento o defecto de las bases creadas en versión 11.2 y superiores de 4D v11 SQL). En este modo, 4D establece "uniones SQL" para las búsquedas por fórmula cuando la fórmula se ajusta para ello (con dos excepciones, ver la descripción del comando **QUERY BY FORMULA** o **QUERY SELECTION BY FORMULA**). **Nota:** si quiere activar las uniones "tipo SQL" (consulte el selector QUERY BY FORMULA Joins selector), siempre debe ejecutar las fórmulas en el servidor de manera que tengan acceso a los registros. Atención, en este contexto, la fórmula no debe contener llamadas a un método, de lo contrario pasará automáticamente al equipo remoto.
HTTP compression level	Entero largo	50	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: no **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
HTTP compression threshold	Entero largo	51	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: no **Valores posibles**: todo valor de tipo entero largo **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Server base process stack size	Entero largo	53	**Alcance**: 4D Server **Se conserva entre dos sesiones**: no **Valores posibles**: entero largo positivo. **Descripción**: tamaño de la pila asignada a cada proceso del sistema preferente en el servidor, expresado en bytes. El tamaño por defecto es determinado por el sistema. Los procesos sistema preferente (procesos de tipo Proceso base 4D client) se cargan para controlar los procesos cliente 4D principales. El tamaño asignado por defecto a la pila de cada proceso preferente da facilidad de ejecución pero puede resultar consecuente cuando se crea un gran número de procesos (varios cientos). Por razones de optimización, este tamaño puede reducirse considerablemente si las operaciones efectuadas por la base lo permiten (por ejemplo si la base no efectúa ordenaciones de grandes cantidades de registros). Son posibles valores de 512 o incluso 256 KB. Sea cuidadoso, subdimensionar la pila es critico y puede afectar la operación de 4D Server. La definición de este parámetro debe hacerse con precaución y tener en cuenta las condiciones de uso de la base (número de registros, tipo de operaciones, etc.). Para que sea tenido en cuenta, este parámetro debe ejecutarse en el equipo servidor (por ejemplo en el **Método base On Server Startup**).

Constante	Tipo	Valor	Comentario
Idle connections timeout	Entero largo	54	**Alcance**: aplicación 4D a menos que valor sea negativo **Se conserva entre dos sesiones:** no **Valores posibles:** valor entero que expresa una duración en segundos. El valor puede ser positivo (nuevas conexiones) o negativo (conexiones existentes). Por defecto, el valor es 20. **Descripción**: máximo periodo de inactividad (timeout) para conexiones al motor de la base 4D y al motor SQL, así como también en modo *ServerNet* (nueva capa de red), al servidor de la aplicación 4D. Cuando una conexión inactiva alcanza este límite, se pone en espera automáticamente, lo cual congela la sesión cliente/servidor y cierra el socket de red. En la ventana de administración del servidor, el estado del proceso del usuario se indica como "Postponed". Este funcionamiento es totalmente transparente para el usuario: tan pronto como hay una nueva actividad en la conexión que está en espera, el socket se reabre automáticamente y la sesión cliente/servidor se restaura. Este parámetro permite, por una parte, economizar los recursos en el servidor: las conexiones en espera cierran el socket y liberan un proceso en el servidor. Por otra parte, esto le permite evitar pérdida de conexiones por el cierre de sockets por parte del firewall. Por esta razón, el valor del timeout para conexiones inactivas deber ser menor que el del firewall en este caso. Si pasa un valor positivo en *valor*, se aplicará a todas las nuevas conexiones en todos los procesos. Si pasa un valor negativo, se aplicará a las conexiones que se abran en el proceso actual. Si pasa 0, las conexiones inactivas no serán sometidas a un timeout. Este parámetro puede definirse del lado del servidor y del cliente. Si pasa dos duraciones diferentes, la más corta se tendrá en cuenta. Por lo general, no necesita cambiar este valor.
PHP interpreter IP address	Entero largo	55	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores**: cadena formateada del tipo "nnn.nnn.nnn.nnn" (por ejemplo "127.0.0.1"). **Descripción**: dirección IP utilizada localmente por 4D para comunicarse con el intérprete PHP vía FastCGI. Por defecto, el valor es "127.0.0.1". Esta dirección debe corresponder a la máquina donde en encuentra 4D. Este parámetro también puede definirse globalmente para todas las máquinas vía las Propiedades de la base. Para mayor información sobre el intérprete PHP, por favor consulte el manual de *Diseño*.
PHP interpreter port	Entero largo	56	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores**: valor de tipo entero largo positivo. Por defecto, el valor es 8002. **Descripción**: número de puerto TCP utilizado o por el intérprete PHP de 4D. Este parámetro también puede modificarse globalmente para todos los equipos vía las Propiedades de la base. Para mayor información sobre el intérprete PHP, consulte el manual de *Diseño*.
PHP number of children	Entero largo	57	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores**: valor de tipo entero largo positivo. Por defecto, el valor es 5. **Descripción**: número de procesos hijos a crear y mantener localmente por el intérprete PHP de 4D. Por razones de optimización, el intérprete PHP crea y utiliza un conjunto (pool) de procesos sistema llamados "procesos hijos" para procesar las peticiones de ejecución de scripts. Puede variar el número de procesos hijo de acuerdo a las necesidades de su aplicación. Este parámetro también puede modificarse globalmente para todos los equipos vía las Propiedades de la base. Para mayor información sobre el intérprete PHP, consulte el manual de *Diseño*. **Nota:** bajo Mac OS, todos los procesos hijos comparten el mismo puerto. Bajo Windows, cada proceso hijo utiliza un número de puerto específico. El primer número es el definido por el intérprete PHP; los otros procesos hijos lo incrementan. Por ejemplo, si el puerto por defecto es 8002 y usted lanza 5 procesos hijos, utilizarán los puertos 8002 a 8006.
PHP max requests	Entero largo	58	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores**: valor de tipo entero largo positivo. Por defecto, el valor es 500. **Descripción**: número máximo de peticiones aceptadas por el intérprete PHP. Cuando se alcanza este número máximo, el intérprete devuelve errores del tipo "servidor ocupado". Por razones de seguridad o rendimiento, puede modificar este valor. Este parámetro también puede modificarse globalmente para todos los equipos vía las Propiedades de la base. Para mayor información sobre este parámetro, consulte la documentación FastCGI-PHP. **Nota:** de parte de 4D, estos parámetros se aplican dinámicamente; no es necesario salir de 4D para que sean tenidos en cuenta. Por otra parte, si el intérprete PHP ya fue lanzado, será necesario salir y lanzarlo nuevamente, para que las modificaciones se tengan en cuenta.
PHP use external interpreter	Entero largo	60	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: no **Valores** : 0 = utilizar intérprete interno, 1 = utilizar intérprete externo **Descripción**: valor que indica si las peticiones PHP de 4D se envían al intérprete interno ofrecido por 4D o a un intérprete externo. Por defecto el valor es 0 (uso del intérprete ofrecido por 4D). Si quiere utilizar su propio intérprete PHP, por ejemplo para beneficiarse de módulos adicionales o de una configuración específica, pase 1 en *valor*. En este caso, 4D no lanza su intérprete interno en caso de peticiones PHP. El intérprete PHP personalizado debe haber sido compilado en FastCGI y estar ubicado en la misma máquina que el motor 4D. Note que en este caso, debe administrar completamente el intérprete; no será iniciado ni detenido por 4D. Este parámetro también puede modificarse globalmente para todas las máquinas vía las Propiedades de la base.

Constante	Tipo	Valor	Comentario
Maximum temporary memory size	Entero largo	61	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: entero largo positivo. **Descripción**: tamaño máximo de memoria temporal que 4D asignar a cada proceso, expresado en MB. Por defecto, el valor es 0 (sin tamaño máximo). 4D utiliza una memoria temporal especial dedicada a las operaciones de indexación y ordenación. Esta memoria conserva la memoria caché "estándar" durante operaciones masivas. Sólo se activa cuando es necesario. Por defecto, el tamaño de la memoria temporal está limitado únicamente por los recursos disponibles (en función de la configuración de memoria del sistema). Este mecanismo es conveniente para la mayoría de las aplicaciones. Sin embargo, en algunos contextos específicos, particularmente cuando una aplicación cliente-servidor efectúa simultáneamente un gran número de ordenaciones secuenciales, el tamaño de la memoria temporal puede aumentar críticamente, hasta volver el sistema inestable. En este contexto, fijar un tamaño máximo para la memoria temporal permite preservar el funcionamiento apropiado de la aplicación. En contraparte, la velocidad de ejecución podría afectarse: cuando se alcanza el tamaño máximo para un proceso, 4D utiliza archivos de discos, que pueden volver lentos los procesos. Para necesidades específicas tales como las descritas anteriormente, un tamaño máximo de 50 MB es generalmente un buen compromiso. Sin embargo, el valor ideal se determinará en función de las especificaciones de la aplicación y será generalmente el resultado de pruebas volumétricas en tiempo real.
SSL cipher list	Cadena	64	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: secuencia de cadenas separadas por dos puntos (por ejemplo "RC4-MD5:RC4-64-MD5:....") **Descripción:** lista de cifrado (*cipher list*) utilizada por 4D para el protocolo seguro. Esta lista modifica la prioridad de los algoritmos de cifrado implementados por 4D. Por ejemplo, puede pasar la siguiente cadena en el parámetro *valor*: "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". Para una descripción completa de la sintaxis para la lista cifrada, consulte la página de cifrado del sitio OpenSSL. Este parámetro es global para la aplicación (concierne al servidor HTTP, al servidor SQL, conexiones cliente/servidor, y también al cliente HTTP y a todos los comandos 4D que usan el protocolo seguro) pero es temporal (no se mantiene entre sesiones). Cuando la lista de cifrado se modifica, debe reiniciar el servidor correspondiente para que los nuevos parámetros sean tenidos en cuenta. Para reinicializar la lista de cifrado a su valor por defecto (guardado permanentemente en el archivo SLI), llame al comando **SET DATABASE PARAMETER** y pase una cadena vacía ("") en el parámetro *valor*. Por defecto, 4D utiliza el algoritmo de cifrado RC4. Si quiere utilizar el algoritmo AES (más reciente), pase la cadena siguiente en el parámetro valor: "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". **Nota:** con el comando **Get database parameter**, la lista de cifrado se devuelve en el parámetro opcional *valorAlfa* y el parámetro de retorno es siempre 0.
Cache unload minimum size	Entero largo	66	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: Entero largo positivo > 1. **Descripción**: tamaño mínimo de memoria a liberar del caché de la base de datos cuando el motor necesita hacer espacio para ubicar un objeto (valor en bytes). El propósito de este selector es reducir el número de liberaciones de datos de la caché con el fin de obtener un mejor rendimiento. Puede hacer variar este parámetro en función del tamaño de la caché y del de los bloques de datos manipulados en su base. Por defecto, si este selector no se utiliza, 4D descarga mínimo 10% de la caché en caso de que se necesite espacio.Alcance: Aplicación 4D Se conserva entre dos sesiones: No Valores posibles: Entero largo positivo > 1. Descripción: tamaño mínimo de memoria a liberar del caché de la base de datos cuando el motor necesita hacer espacio para ubicar un objeto (valor en bytes). El propósito de este selector es reducir el número de liberaciones de datos de la caché con el fin de obtener un mejor rendimiento. Puede hacer variar este parámetro en función del tamaño de la caché y del de los bloques de datos manipulados en su base. Por defecto, si este selector no se utiliza, 4D descarga mínimo 10% de la caché en caso de que se necesite espacio.
Direct2D status	Entero largo	69	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Descripción**: modo de activación de Direct2D bajo Windows. **Valores posibles:** una de las siguientes constantes (modo 3 por defecto): Direct2D Disabled (0): el modo Direct2D no está activo, la base funciona en el modo anterior (GDI/GDIPlus). Direct2D Hardware (1): uso de Direct2D en contexto de hardware gráfico en toda la aplicación 4D. Si este contexto no está disponible, uso del contexto de software gráfico Direct2D (excepto bajo Vista, en cuyo caso el modo GDI/GDIPlus se utiliza para un mejor rendimiento). Direct2D Software (3) (Modo por defecto): a partir de Windows 7, uso de Direct2D en contexto de software gráfico en toda la aplicación 4D. En Vista, por razones de rendimiento se utiliza el modo GDI/GDIPlus. **Nota de compatibilidad:** a partir de 4D v14, los modos híbridos se desactivan y redireccionan a los modos disponibles (el antiguo modo 2 es equivalente a 1; los antiguos modos 4 y 5 son equivalentes al modo 3).

Constante	Tipo	Valor	Comentario

Direct2D get active status	Entero largo	74	**Nota**: sólo puede utilizar este selector con el comando **Get database parameter** y su valor no puede definirse.
			Descripción: devuelve la implementación activa de Direct2D bajo Windows.
			Valores posibles: 0, 1, 2, 3, 4 o 5 (ver los valores del selector 69). El valor devuelto depende de la disponibilidad de Direct2D, del hardware y de la calidad Direct2D soportado por el sistema operativo.
			Por ejemplo, si ejecuta:
			`SET DATABASE PARAMETER(Direct2D status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D get active status)`
			- En Windows 7 y superiores, $mode vale 1 cuando el sistema detecta un hardware compatible con Direct2D; de lo contrario, $mode valdrá 3 (contexto software).
			- En Windows Vista, $mode valdrá 1 si el sistema detecta un hardware compatible con Direct2D; de lo contrario, $mode toma el valor 0 (desactivando Direct2D).
			- En Windows XP, $mode siempre valdrá 0 (no compatible con Direct2D).

Diagnostic log recording	Entero largo	79	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: 0 ó 1 (0 = no guardar,1 = guardar) **Descripción**: inicio o detención del registro del archivo de diagnóstico de 4D. Por defecto, el valor es 0 (no guarda). 4D permite guardar de manera continua en un archivo de diagnóstico un conjunto de eventos relativos al funcionamiento interno de la aplicación. La información contenida en este archivo está destinada a la actualización de las aplicaciones 4D y puede ser analizada con ayuda de los servicios técnicos de 4D. Cuando pasa 1 en este selector, el archivo de diagnóstico, llamado *NomBase.txt*, se crea automáticamente (o abre) en la carpeta **Logs** de la base. Una vez el archivo alcance un tamaño de 10 MB, se cierra y se genera un nuevo archivo *NomBase_N.txt*, con un número secuencial N incrementado. Note que es posible incluir la información personalizada en este archivo con ayuda del comando **LOG EVENT**.
Log command list	Cadena	80	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: cadena que contiene la lista de números de los comandos 4D a guardar (separados por dos puntos), "all" para guardar todos los comandos o "" (cadena vacía) para no guardar ninguno. **Descripción**: la lista de comandos 4D a guardar en el archivo de depuración (ver el selector 34, Debug Log Recording). Por defecto, se guardan todos los comandos 4D. Este selector permite guardar la cantidad de información almacenada en el archivo de depuración limitando los comandos 4D donde quiera guardar la ejecución.
Spellchecker	Entero largo	81	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: 0 (por defecto) = corrector macOS nativo (Hunspell desactivado), 1 = corrector Hunspell activo. **Descripción**: permite activar el corrector ortográfico Hunspell bajo macOS. Por defecto, en esta plataforma el corrector nativo está activo. Puede preferir utilizar el corrector Hunspell, por ejemplo, para unificar la interfaz de sus aplicaciones multiplataformas (bajo Windows, sólo el corrector Hunspell está disponible). Para mayor información, consulte **Corrección ortográfica**.
QuickTime support	Entero largo	82	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: Sí **Valores posibles**: 0 (por defecto) = QuickTime desactivado, 1 = QuickTime activado. **Descripción**:en 4D a partir de la v14, por defecto los codecs QuickTime ya no se soportan. Por compatibilidad, puede utilizar este selector para reactivarlos en su base. La modificación de esta opción requiere que la base se reinicie. Sin embargo, debe notar que en futuras versiones de 4D, se eliminará de forma permanente el soporte QuickTime.

Constante	Tipo	Valor	Comentario
Dates inside objects	Entero largo	85	**Alcance**: Proceso actual **Se conserva entre dos sesiones:** No **Valores posibles**: String type without time zone (0), String type with time zone (1), Date type (2) (por defecto) **Descripción**: define la forma en que se almacenan las fechas dentro de los objetos, así como también cómo se importan / exportan en JSON. Cuando el valor del selector es Date type (valor predeterminado para las bases creadas con 4D v17 y superior), las fechas 4D se almacenan con el tipo de fecha dentro de los objetos, con respecto a la configuración de fecha local. Cuando se convierte a formato JSON, los atributos de fecha se convertirán en cadenas que no incluyen un tiempo. (**Nota:** esta configuración se puede definir mediante la opción "Utilizar tipo de fecha en lugar del formato de fecha ISO en objetos" que se encuentra en **Página Compatibilidad** de la configuración de la base). Si pasa String type with time zone en este selector, convertirá las fechas 4D en cadenas ISO y tendrá en cuenta la zona horaria local. Por ejemplo, la conversión de la fecha 23/08/2013 le da "2013-08-22T22: 00: 000Z" en formato JSON cuando la operación se realiza en Francia durante el horario de verano (GMT+ 2). Este principio se ajusta al funcionamiento estándar de JavaScript. Esto puede ser una fuente de errores cuando desea enviar valores de fecha JSON a alguien en un huso horario diferente. Por ejemplo, cuando exporta una tabla usando **Selection to JSON** en Francia que se debe reimportar en los EE. UU. utilizando **JSON TO SELECTION**. Dado que las fechas se vuelven a interpretar en cada zona horaria, los valores almacenados en la base de datos serán diferentes. En este caso, puede modificar el modo de conversión de las fechas para que no tengan en cuenta la zona horaria pasando String type without time zone en este selector. La conversión de la fecha 23/08/2013 le dará "2013-08-23T00: 00: 00Z" en todos los casos.
Use legacy network layer	Entero largo	87	**Alcance:** 4D en modo local, 4D Server **Se conserva entre dos sesiones:** sí **Descripción:** fija u obtiene el estado actual de la capa de red antigua para las conexiones cliente/servidor. La capa de red antigua es obsoleta a partir de 4D v14 R5 y debe ser reemplazada progresivamente en sus aplicaciones por la capa de red *ServerNet*. *ServerNet* será requerida en próximas versiones 4D con el fin de beneficiarse de las futuras evoluciones de la red. Por razones de compatibilidad, la capa de red antigua aún se soporta para permitir una transición sin problemas para las aplicaciones existentes; (se usa por defecto en aplicaciones convertidas de una versión anterior a v14 R5). Pase 1 en este parámetro para utilizar la capa de red antigua (y desactivar *ServerNet*) para las conexiones cliente/servidor, y pase 0 para deshabilitar la red antigua (y utilizar *ServerNet*). Esta propiedad también se puede definir mediante la opción "Usar capa de red antigua " que se encuentran en **Página Compatibilidad** de las Propiedades de la base (ver **Opciones red y cliente-servidor**). En esta sección, también puede encontrar una discusión sobre la estrategia de migración. Le recomendamos que active *ServerNet* tan pronto como sea posible. Deberá reiniciar la aplicación para que este parámetro sea tenido en cuenta. No está disponible en 4D Server v14 R5 64-bit versión para OS X, que sólo soporta el *ServetNet*; (siempre devuelve 0). **Valores posibles:** 0 o 1 (0 = no utilizan capa de red antigua, 1 = uso capa de red antigua) **Valor por defecto:** 0 en bases de datos creadas con 4D v14 R5 o superior, 1 en bases de datos convertidas de 4D v14 R4 o anteriores.
SQL Server Port ID	Entero largo	88	**Alcance**: 4D modo local y 4D Server. **Se conserva entre dos sesiones**: Sí **Descripción**: permite leer o definir el número del puerto TCP utilizado por el servidor SQL integrado de 4D en modo local o 4D Server. Por defecto, el valor es 19812. Cuando se define este selector, la configuración de la base se actualiza. También puede definir el número del puerto TCP en la página "SQL" de la caja de diálogo de Propiedades de la base. **Valores posibles:** 0 a 65535. **Valor por defecto:** 19812
Circular log limitation	Entero largo	90	**Alcance**: 4D local, 4D Server. **Se conserva entre dos sesiones:** no **Valores posibles**: todo valor entero, 0 = conservar todos los registros **Descripción**: número máximo de archivos a conservar en rotación para cada tipo de registro. Por defecto, todos los archivos se conservan. Si pasa un valor *X*, solo los *X* archivos más recientes se conservan, el más antiguo se borra automáticamente cuando se crea uno nuevo. Este ajuste se aplica a cada uno de los siguientes archivos de registro: registros de peticiones (selectores 28 y 45), registro de depuración (selector 34), registro de eventos (selector 79), así como el historial de peticiones web y el historial de depuración Web (selectores 29 y 84 del comando **WEB SET OPTION**).
Number of formulas in cache	Entero largo	92	**Alcance**: aplicación 4D **Se conserva entre dos sesiones:** no **Valores posibles**: enteros largos positivos **Valor por defecto**: 0 (sin caché) **Descripción**: establece u obtiene el número máximo de fórmulas a conservar en la memoria caché de fórmulas, que es utilizado por el comando **EXECUTE FORMULA**. Este límite se aplica a todos los procesos, pero cada proceso tiene su propia caché de fórmulas. Ubicar las fórmulas en la caché acelera la ejecución del comando **EXECUTE FORMULA** en modo compilado, ya que cada fórmula en caché se tokeniza sólo una vez en este caso.Cuando se cambia el valor de la memoria caché, el contenido existente se restablecen incluso si el nuevo tamaño es más grande que el anterior. Una vez se alcanza el número máximo de fórmulas en la memoria caché, una nueva fórmula ejecutada borrará a la más antigua de la memoria caché (modo FIFO). Este parámetro sólo se tiene en cuenta en las bases o componentes compilados.

Constante	Tipo	Valor	Comentario
Cache flush periodicity	Entero largo	95	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones:** no **Valores posibles:** entero largo > 1 (segundos) **Descripción**: obtiene o establece la periodicidad del vaciado de la caché, expresado en segundos. La modificación de este valor prevalece sobre la opción **Vaciar caché cada X segundos** en **Página Base de datos/Memoria** de la configuración de la base para la sesión (que no se almacena en las Propiedades de la base).
Tips enabled	Entero largo	101	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: 0 = consejos desactivados, 1 = consejos activados (predeterminado) **Descripción**: define u obtiene el estado de visualización actual de los consejos para la aplicación 4D. De forma predeterminada, las sugerencias están activadas. Tenga en cuenta que este parámetro define todos los consejos 4D, es decir, los mensajes de ayuda de formulario y las sugerencias del editor de modo Diseño.
Tips delay	Entero largo	102	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: entero largo >= 0 (tics) **Descripción**: retraso antes de que se muestren las sugerencias una vez que el cursor del ratón se haya detenido en objetos con mensajes de ayuda adjuntos. El valor se expresa en tics (1/60 de segundo). El valor predeterminado es 45 tics (0.75 segundos).
Tips duration	Entero largo	103	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: entero largo >= 60 (tics) **Descripción**: duración máxima de visualización de una sugerencia. El valor se expresa en tics (1/60 de segundo). El valor predeterminado es 720 tics (12 segundos).
Min TLS version	Entero largo	105	**Alcance**: 4D Server, 4D Web Server y 4D SQL Server **Conservar entre dos sesiones**: No **Descripción**: se utiliza para especificar el nivel TLS (Transport Layer Security), que ofrece cifrado y autenticación de datos entre aplicaciones y servidores. Los valores definidos se aplican globalmente a la capa de red. Una vez modificado, el servidor debe reiniciarse para utilizar el nuevo valor. El protocolo mínimo predeterminado es TLSv1_2. Se rechazarán los intentos de conexión de clientes con un valor inferior al mínimo. **Valores posibles**: • TLSv1_0 - TLS 1.0, introducido en 1999 como una actualización de SSL v3.0. TLS 1.0 y SSL 3.0 no interoperan. • TLSv1_1 - TLS 1.1, introducido en 2006 como una actualización de TLS 1.0. Las mejoras incluyen mejor seguridad y manejo de errores, etc. • TLSv1_2 - TLS 1.2, introducido en 2008 como una actualización de TLS 1.1. Las mejoras incluyen una mayor flexibilidad, soporte adicional para el cifrado autenticado, etc. **NOTAS**: - El plugin 4D Internet Commands utiliza una capa de red diferente, por lo que este selector no tendrá ningún impacto en su versión TLS. - Se ignorarán los intentos de aplicar TLS a la capa de red heredada.
Times inside objects	Entero largo	109	**Alcance**: 4D local, 4D Server (todos los procesos) **Se conserva entre dos sesiones**: No **Valores posibles**: Times in seconds (0) (predeterminado), Times in milliseconds (1) **Descripción**: define la forma en que los valores de tipo hora se convierten y almacenan dentro de las propiedades de los objetos y los elementos de la colección, así como la forma en que se importan/exportan en JSON y en las áreas web. Por defecto, a partir de 4D v17, las horas se convierten y almacenan en número de segundos en los objetos. En versiones anteriores, los valores de tiempo se convertían y almacenaban como cantidad de milisegundos en esos contextos. Usar este selector puede ayudarlo a migrar sus aplicaciones volviendo a la configuración anterior si es necesario. **Nota**: los métodos ORDA y el motor SQL ignoran esta configuración, siempre suponen que los valores de tiempo son números de segundos.

Nota: el parámetro *tabla* sólo es utilizado por los selectores 31, 32, 46 y 47. En todos los demás casos, se ignora si se pasa.

Si no se mantiene una configuración constante entre sesiones, pero desea asegurarse de que se aplique, debe ejecutarla en [#title id="142"/] o **Método base On Server Startup**.

Ejemplo 1

La siguiente instrucción evitará un posible problema de timeout:

```
  `Aumento del timeout a 3 horas para el proceso actual
SET DATABASE PARAMETER(4D Server Timeout;-60*3)
  `Ejecución de una operación larga sin control de 4D
...
WR PRINT MERGE(Area;3;0)
...
```

Ejemplo 2

Este ejemplo forza temporalmente la ejecución de un comando búsqueda por fórmula en el equipo cliente:

```
curVal:=Get database parameter([tabla1];Query By Formula On Server)  `Almacena la configuración actual
SET DATABASE PARAMETER([tabla1];Query By Formula On Server;1)  `Fuerza la ejecución en el equipo cliente
```

Ejemplo 3

Usted quiere exportar datos en JSON que contienen una fecha 4D convertida. Note que la conversión ocurre cuando la fecha se guarda en el objeto, debe llamar al comando **SET DATABASE PARAMETER** antes de llamar a **OB SET**:

```
C_OBJECT($o)
SET DATABASE PARAMETER(Dates inside objects;0)
OB SET($o ;"myDate";Current date)  // conversión JSON
$json:=JSON Stringify($o)
SET DATABASE PARAMETER(Dates inside objects;1)
```

SET UPDATE FOLDER

SET UPDATE FOLDER (rutaCarpeta {; erroresDiscretos})

Parámetro	Tipo		Descripción
rutaCarpeta	Cadena	➡	Ruta de acceso de la carpeta (paquete bajo OS X) que contiene la aplicación actualizada
erroresDiscretos	Booleano	➡	False (por defecto) = mostrar mensajes de error, True = no reportarlos

Descripción

El comando **SET UPDATE FOLDER** especifica la carpeta que contiene la actualización de la aplicación 4D fusionada actual. Esta información se almacena en la sesión 4D hasta que se llama el método **RESTART 4D**. Si se sale de la aplicación manualmente, esta información no se conserva.

Este comando está destinado a ser utilizado en un proceso de actualización automática de una aplicación fusionada (servidor o monopuesto). Para más información, consulte el capítulo **Terminar y desplegar aplicaciones finales** en el Manual de *Diseño*.

Nota: este comando sólo funciona con 4D Server o con una aplicación monopuesto fusionada con 4D Volume Desktop.
En el parámetro *rutaCarpeta*, pase la ruta de acceso completa de la carpeta de la nueva versión de la aplicación fusionada (carpeta que contiene la aplicación *my4DApp.exe* bajo Windows o el paquete *my4DApp.app* bajo OS X), creado por el generador de aplicaciones de 4D. La nueva versión debe haber sido generada por el generador de aplicaciones de 4D v14. En particular, debe contener una versión actualizada de la herramienta "updater" incluida en 4D y que se utiliza para administrar las actualizaciones remotas.

Nota: le recomendamos que utilice los mismos nombres de la versión original para los archivos de la nueva versión de la aplicación, ya que la carpeta de la aplicación se sustituye durante la actualización. Si utiliza nombres diferentes para estos archivos, los atajos y/o rutas almacenados ya no funcionarán más.

Si los parámetros son válidos, la actualización se pone "en espera" en la sesión hasta que se llame el comando **RESTART 4D**. Si ha ejecutado varias veces **SET UPDATE FOLDER** antes de llamar a [**RESTART 4D**, se tiene en cuenta la última llamada válida.

Puede pasar una cadena vacía ("") en el parámetro *rutaCarpeta* para reinicializar la información de actualización para la sesión actual.

El parámetro opcional *erroresDiscretos* especifica cómo se reportan los errores por la herramienta "updater":

- si pasa **False** o si se omite este parámetro, los errores se registran en el historial de actualización y se muestran en una caja de diálogo de alerta.
- si pasa **True**, los errores sólo se registran en el historial de actualización.

Excepción: si la herramienta "updater" no puede crear el archivo de historial, se muestra una caja de diálogo de alerta, independientemente del valor del parámetro *erroresDiscretos*. Para más información, consulte la descripción del comando **Get last update log path**.

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1, de lo contrario, toma el valor 0. Puede interceptar los errores generados por el comando utilizando un método instalado utilizando el comando **ON ERR CALL**.

Ejemplo

Usted creó una carpeta "MyUpdates" en su disco, en la cual ubicó una nueva versión de la aplicación "MyApp". Usted no desea mostrar los errores. Para preparar la actualización, escribe:

```
// Sintaxis Windows
SET UPDATE FOLDER("C:\\MyUpdates"+Folder separator+"MyApp"+Folder separator;True)

// Sintaxis OS X
SET UPDATE FOLDER("MacHD:MyUpdates"+Folder separator+"MyApp.app"+Folder separator;True)
```

Structure file

Structure file {(*)} -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Devuelve el archivo de estructura de la base local
Resultado	Cadena	➲	Nombre largo del archivo de estructura de la base

Descripción

El comando **Structure file** devuelve el nombre largo del archivo de estructura de la base en la cual está trabajando actualmente.

En Windows
Si, por ejemplo, está trabajando con la base MisCDs ubicada en \DOCS\MisCDs en el disco G, el comando devuelve G:\DOCS\MisCDs\MisCDs.4DB.

En Macintosh
Si, por ejemplo, está trabajando, con la base ubicada en la carpeta Documentos:MisCDsƒ: en el disco Macintosh HD, el comando devuelve Macintosh HD:Documentos:MisCDsƒ:MisCDs.

Nota: en el caso particular de una base compilada y fusionada con 4D Volume Desktop, este comando devuelve la ruta de acceso al archivo de la aplicación (archivo ejecutable) bajo Windows y OS X. Bajo OS X, este archivo está ubicado dentro del paquete del programa, en la carpeta [Contents:Mac OS]. Este funcionamiento viene de un antiguo mecanismo y se conserva por razones de compatibilidad. Si quiere obtener el nombre completo del paquete del programa, es preferible utilizar el comando **Application file**. La técnica consiste en probar la aplicación utilizando el comando **Application type**, luego se ejecuta **Structure file** o **Application file** dependiendo del contexto.

Advertencia: si llama este comando mientras utiliza 4D en modo remoto, sólo devuelve el nombre del archivo de estructura; no el nombre largo.

El parámetro opcional * es útil en el caso de una arquitectura que utilice componentes: permite determinar la estructura (local o componente) para la cual usted quiere obtener el nombre largo en función del contexto en el cual se llama el comando:

- Cuando se llama el comando desde un componente:

  - Si se pasa el parámetro *, el comando devuelve el nombre largo del archivo de estructura de la base local,

  - Si no se pasa el parámetro *, el comando devuelve el nombre largo del archivo de estructura del componente.

El archivo de estructura del componente corresponde al archivo .4db o .4dc del componente que se encuentra en la carpeta "Components" de la base. Sin embargo, un componente también puede instalarse como un alias/atajo o un carpeta/paquete .4dbase:

- En el caso de un componente instalado en forma de alias/atajo, el comando devuelve la ruta de acceso del archivo .4db o .4dc original (el alias o atajo es resuelto).

- En el caso de un componente instalado como una carpeta/paquete .4dbase, el comando devuelve la ruta de acceso del archivo .4db o .4dc al interior de esta carpeta/paquete.

• Cuando el comando se llama desde un método de la base local, siempre devuelve el nombre largo del archivo de estructura de la base local, sin importar si se pasa o no el parámetro *.

Ejemplo 1

Este ejemplo muestra el nombre y la ubicación del archivo de estructura que está utilizando:

```
If(Application type#4D Remote mode)
   $vsStructureFilename:=Long name to file name(Structure file)
   $vsStructurePathname:=Long name to path name(Structure file)
   ALERT("You are currently using the database "+Char(34)+$vsStructureFilename+Char(34)+
   " located at "+Char(34)+$vsStructurePathname+Char(34)+".")
Else
   ALERT("You are connected to the database "+Char(34)+Structure file+Char(34))
End if
```

Nota: los métodos de proyecto **Long name to file name** y **Long name to path name** se describen en detalle en la sección **Documentos del sistema**.

Ejemplo 2

El siguiente ejemplo puede utilizarse para saber si el método se llama desde un componente:

```
C_BOOLEAN($0)
$0:=(Structure file#Structure file(*))
  ` $0=True si el método es llamado desde un componente
```

VERIFY CURRENT DATA FILE

VERIFY CURRENT DATA FILE {(objetos ; opciones ; metodo {; arrayTablas {; arrayCampos}})}

Parámetro	Tipo		Descripción
objetos	Entero largo	➡	Objetos a verificar
opciones	Entero largo	➡	Opciones de verificación
metodo	Texto	➡	Nombre del método 4D de retrollamada
arrayTablas	Array entero largo	➡	Números de las tablas a verificar
arrayCampos	Array entero 2D, Array entero largo 2D, Array real 2D	➡	Números de los índices a verificar

Descripción

El comando **VERIFY CURRENT DATA FILE** efectúa una verificación estructural de los objetos encontrados en el archivo de datos abierto actualmente por 4D.

Este comando tiene un funcionamiento idéntico al del comando **VERIFY DATA FILE**, excepto que sólo aplica al archivo de datos actual de la base de datos abierta. No son necesarios los parámetros que especifican la estructura y los datos.
Consulte el comando **VERIFY DATA FILE** para la descripción de los parámetros.

Si pasa el comando **VERIFY CURRENT DATA FILE** sin parámetros, la verificación se lleva a cabo con los valores por defecto de los parámetros:

- *objetos* = Verificar todos (= valor 16)
- *opciones* = 0 (se crea el archivo de historial pero sin marca de tiempo)
- *metodo* = ""
- *arrayTablas* y *arrayCampos* se omiten.

Cuando se ejecuta este comando, el caché de los datos se vacía y todas las operaciones de acceso de datos se bloquean durante la verificación.

Si un archivo de historial se ha generado, su ruta completa se devuelve en la variable sistema *Document*.

Variables y conjuntos del sistema

Si el método de retrollamada no existe, la verificación no se efectúa, se genera un error y la variable sistema OK toma el valor 0.
Si un archivo de historial se ha generado, su ruta completa se devuelve en la variable sistema Document.

VERIFY DATA FILE

VERIFY DATA FILE (rutaEstructura ; rutaDatos ; objetos ; opciones ; metodo {; arrayTablas {; arrayCampos}})

Parámetro	Tipo		Descripción
rutaEstructura	Texto	➡	Ruta de acceso al archivo de estructura de la base a verificar
rutaDatos	Texto	➡	Ruta de acceso del archivo de datos de la base a verificar
objetos	Entero largo	➡	Objetos a verificar
opciones	Entero largo	➡	Opciones de verificación
metodo	Texto	➡	Nombre del método 4D de retrollamada
arrayTablas	Array entero largo	➡	Números de las tablas a verificar
arrayCampos	Array entero 2D, Array entero largo 2D, Array real 2D	➡	Números de los índices a verificar

Descripción

El comando **VERIFY DATA FILE** efectúa una verificación estructural de los objetos contenidos en el archivo de datos 4D designado por *rutaEstructura* y *rutaDatos*.

Nota: para mayor información sobre el proceso de verificación de datos, consulte el Manual de Diseño. *rutaEstructura* designa el archivo de estructura (compilado o no) asociado con el archivo de datos a verificar. Puede tratarse del archivo de estructura abierto o de cualquier otro archivo de estructura. Usted debe pasar un nombre de ruta completo, expresado con la sintaxis del sistema operativo. También puede pasar una cadena vacía, en este caso aparece una caja de diálogo estándar de apertura de archivos que permite al usuario designar el archivo de estructura a utilizar.
rutaDatos designa un archivo de datos 4D (.4DD). Debe corresponder al archivo de estructura definido por el parámetro *rutaEstructura*. Atención, usted puede designar el archivo de estructura actual pero el archivo de datos no debe ser el archivo actual (abierto). Para verificar el archivo de datos está abierto actualmente, utilice el comando **VERIFY CURRENT DATA FILE**. Si intenta verificar el archivo de datos actual con el comando **VERIFY DATA FILE**, se genera un error.

El archivo de datos designado se abre en modo sólo lectura. Debe asegurarse de que ninguna aplicación acceda a este archivo en modo escritura, de lo contrario los resultados de la verificación podrían ser distorsionados.

En el parámetro *rutaDatos*, puede pasar una cadena vacía, un nombre de archivo o una ruta de acceso completa, expresada en la sintaxis del sistema operativo. Si pasa una cadena vacía, aparecerá la caja de diálogo estándar de apertura de archivos de manera que el usuario puede especificar el archivo a revisar (note que en este caso, no es posible seleccionar el archivo de datos actual). Si pasa únicamente un nombre de archivo de datos, 4D lo buscará en el mismo nivel que el archivo de estructura especificado.

El parámetro *objetos* se utiliza para designar los tipos de objetos a verificar. Puede verificar dos tipos de objetos: registros e índices. Puede utilizar las siguientes constantes, que se encuentran en el tema "**Mantenimiento archivo de datos**":

Constante	Tipo	Valor	Comentario
Verify all	Entero largo	16	
Verify indexes	Entero largo	8	Esta opción verifica la consistencia física de los índices, sin enlace a los datos. Señala llaves inválidas pero no le permite detectar llaves duplicadas (dos índices que apuntan al mismo registro). Este tipo de error sólo puede detectarse con la opción Verificar todos.
Verify records	Entero largo	4	

Para verificar los registros y los índices, pase el total de Verify Records+Verify Indexes. El valor 0 (cero) también puede ser utilizado para obtener el mismo resultado. La opción Verify All realiza una verificación interna completa. Esta verificación es compatible con la creación de un historial.

El parámetro *opciones* se utiliza para definir las opciones de verificación. Las siguientes opciones están disponibles, accesibles vía las constantes del tema **Mantenimiento archivo de datos**:

Constante	Tipo	Valor	Comentario
Do not create log file	Entero largo	16384	Por lo general, este comando crea un archivo de historial en formato XML (consulte el final de la descripción del comando). Con esta opción, no se creará un archivo de historial.
Timestamp log file name	Entero largo	262144	Cuando esta opción se pasa, el nombre del archivo de historial generado contendrá la fecha y hora de su creación; como resultado, no reemplazará cualquier archivo de historial generado anteriormente. Por defecto, si no se pasa esta opción, los nombres de archivos de historial no son marcados con la fecha y hora y cada nuevo archivo generado sustituye al anterior.

En principio, el comando **VERIFY DATA FILE** crea un archivo de historial en formato XML (por favor vaya al final de la descripción de este comando). Puede cancelar esta operación pasando esta opción. Para crear el archivo de historial, pase 0 en *opciones*.
El parámetro *metodo* permite definir un método de retrollamada que será llamado regularmente durante la verificación. Si pasa una cadena vacía, no se llama ningún método. Si el método pasado no existe, la verificación no se lleva a cabo, se genera un error y la variable OK toma el valor 0. Cuando se llama, este método recibe hasta 5 parámetros en función de los objetos verificados y de tipo de evento que origina la llamada (ver la tabla de llamadas). Es imperativo declarar estos parámetros en el método:

- $1 Entero largo Tipo de mensaje (ver tabla)
- $2 Entero largo Tipo de objeto
- $3 Text Mensaje
- $4 Entero largo Número de tabla
- $5 Entero largo Reservado

La siguiente tabla describe el contenido de los parámetros en función del tipo de evento:

Evento	$1 (Entero largo)	$2 (Entero largo)	$3 (Texto)	$4 (Entero largo)	$5 (Entero largo)
Mensaje	1	0	Mensaje de progresión	Porcentaje realizado	Reservado
Fin de la verificación (**)	2	Tipo de objeto (**)	Texto del mensaje OK	Número de tabla o de índice	Reservado
Error	3	Tipo de objeto (**)	Texto dek mensaje de error	Número de tabla o de índice	Reservado
Fin de ejecución	4	0	DONE	0	Reservado
Advertencia	5	Tipo de objeto (**)	Texto dek mensaje de error	Número de tabla o de índice	Reservado

(*) El evento *Fin de la verificación* ($2=1) no se devuelve nunca cuando el modo de verificación es Verify All. Sólo se utiliza en modo Verify Records o Verify Indexes.
(**) *Tipo de objeto*: cuando un objeto se verifica, puede enviarse un mensaje "terminado" ($1=2), error ($1=3) o terminado ($1=5). El tipo de objeto devuelto en $2 puede ser uno de los siguientes:

- 0 = indeterminado
- 4 = registro
- 8 = índice
- 16 = objeto estructura (control preliminar del archivo de datos).

Caso particular: cuando $4 = 0 para $1=2, 3 ó 5, el mensaje no concierne a una tabla sino a un archivo de datos en su conjunto.
El método de retrollamada también debe retornar un valor en $0 (Entero largo), permitiendo controlar la ejecución de la operación:

- Si $0 = 0, la operación continúa normalmente
- Si $0 = -128, la operación se detiene sin que se genere error
- Si $0 = otro valor, la operación se detiene y el valor pasado en $0 se devuelve como número de error. Este error puede ser interceptado por un método de gestión de errores.

Nota: no es posible interrumpir la ejecución vía $0 luego de que el evento se haya generado *Fin de ejecución* ($4=1).
Dos arrays opcionales también pueden ser utilizados por este comando:

- El array *arrayTablas* contiene los números de las tablas cuyos registros van a ser verificados. Permite limitar la verificación de ciertas tablas. Si este parámetro no se pasa o si el array está vacío y el parámetro *objetos* contiene Verify Records, todas las tablas serán verificadas.
- El *arrayCampos* contiene los números de los campos indexados que deben ser verificados. Si este parámetro no se pasa o si el array está vacío y el parámetro *objetos* contiene Verify Indexes, todos los índices serán verificados. El comando ignora los campos que no están indexados. Si un campo contiene varios índices, todos son verificados. Si un campo forma parte de un índice compuesto, la totalidad del índice se verifica.
  Debe pasar un array 2D en *arrayCampos*. Para cada línea del array:
  - El elemento {0} contiene el número de la tabla,
  - Los otros elementos {1...x} contienen los números de los campos.

Por defecto, el comando **VERIFY DATA FILE** crea un archivo de historial en formato XML (si no ha pasado la opción Do not create log file, vea el parámetro *opciones*). Su nombre está basado en el archivo de estructura de la base actual y está ubicado en la carpeta **Logs** de esta base. Por ejemplo, para un archivo de estructura llamado "myDB.4db," el archivo de historial se llamará "myDB_Verify_Log.xml."
Si ha pasado la opción Timestamp log file name, el nombre del archivo de historial incluye la fecha y la hora de su creación en la forma "AAAA-MM-DD HH-MM-SS", que nos da, por ejemplo: "myDB_Verify_Log_2015-09-27 15-20-35.xml". Esto significa que cada nuevo archivo de historial no reemplaza al anterior, pero podría requerir acción manual posterior para eliminar archivos innecesarios.
Independientemente de la opción seleccionada, tan pronto como se genera un archivo de historial, su ruta se devuelve en la variable del sistema *Document* después de la ejecución del comando.

Ejemplo 1

Verificación simple de los datos y de los índices:

```
VERIFY DATA FILE($NomEstructura;$NomData;Verify indexes+Verify records;Do not create log file;"")
```

Ejemplo 2

Verificación completa con archivo de historial:

```
VERIFY DATA FILE($NomEstructura;$NomData;Verify All No Callback;0;"")
```

Ejemplo 3

Verificación de los registros únicamente:

```
VERIFY DATA FILE($NomEstructura;$NomData;Verify records;0;"")
```

Ejemplo 4

Verificación de los registros de las tablas 3 y 7 únicamente:

```
ARRAY LONGINT($arrTableNums;2)
$arrTableNums{1}:=3
$arrTableNums{2}:=7
VERIFY DATA FILE($StructName;$DataName;Verify records;0;"FollowScan";$arrTableNums)
```

Ejemplo 5

Verificación de índices específicos (índice del campo 1 de la tabla 4 e índice de los campos 2 y 3 de la tabla 5):

```
ARRAY LONGINT($arrTablaNums;0)  `no utilizado pero obligatorio
ARRAY LONGINT($arrIndex;2;0)  `2 líneas (columnas añadidas luego)
$arrIndex{1}{0}:=4  ` número de tabla en elemento 0
APPEND TO ARRAY($arrIndex{1};1)  `número del primer campo a verificar
$arrIndex{2}{0}:=5  ` número de la tabla en elemento 0
APPEND TO ARRAY($arrIndex{2};2)  ` número del primer campo a verificar
APPEND TO ARRAY($arrIndex{2};3)  ` número del segundo campo a verificar
VERIFY DATA FILE($NomEstructura;$NomData;Verify indexes;0;"FollowScan";$arrTablaNums;$arrIndex)
```

Variables y conjuntos del sistema

Si el método de retrollamada no existe, la verificación no se efectúa, se genera un error y la variable sistema OK toma el valor 0.
Si un archivo de historial se ha generado, su ruta completa se devuelve en la variable sistema Document.

Version type

Version type -> Resultado

Parámetro	Tipo	Descripción
Resultado	Entero largo	Versión de demostración o estándar, versión 64 bits o 32 bits, base 4D o aplicación fusionada

Descripción

El comando **Version type** devuelve un valor numérico que representa el tipo de versión de 4D o de 4D Server que está utilizando. 4D ofrece las siguientes constantes predefinidas, que se encuentran en el tema **Entorno 4D**:

Constante	Tipo	Valor	Comentario
64 bit version	Entero largo	1	
Demo version	Entero largo	0	
Merged application	Entero largo	2	La version es una aplicación fusionada con 4D Volume Desktop

Nota: en las versiones actuales de 4D, el modo demostración no está disponible.
Version type devuelve un valor en forma de un *campo de bits*; es necesario utilizar los operadores de bits para interpretarlo (ver los ejemplos).

NOTA DE COMPATIBILIDAD: en versiones de 4D anteriores a la 13.2, un conjunto de constantes diferentes estaba disponible para este comando; sin embargo, estas constantes no manejan algunos casos correctamente, por esta razón fueron modificadas. Esta modificación significa que su código debe adaptarse (ver ejemplo). Sin embargo, si quiere conservar el funcionamiento anterior, puede remplazar las constantes en su código existente por sus valores anteriores: 2 para versión 64 bits, 1 para versión de demostración, 0 para versión estándar.

Ejemplo 1

Su aplicación 4D contiene código específico en función de la versión en ejecución. Puede conocer el entorno de ejecución utilizando el siguiente código:

```
If(Version type?? 64 bit Version)
  // Estamos en una versión 64 bits
Else
  // Estamos en una versión 32 bits
End if
```

Ejemplo 2

Esta prueba permite ejecutar código diferente dependiendo de que versión es una aplicación fusionada o una base abierta por 4D/4D Server:

```
If(Version type?? Merged application)
  // Estamos en una aplicación fusionada
Else
  // Estamos en una base ejecutada por 4D
End if
```

_o_ADD DATA SEGMENT

_o_ADD DATA SEGMENT
Este comando no requiere parámetros

Descripción

Nota de compatibilidad: a partir de la versión 11 de 4D, no se soportan los segmentos de datos (el tamaño del archivo de datos ahora es ilimitado).

_o_DATA SEGMENT LIST

_o_DATA SEGMENT LIST (Segmentos)

Parámetro	Tipo		Descripción
Segmentos	Array cadena	⇦	Nombres largos de los segmentos de datos de la base

Nota de compatibilidad

Desde la versión 11 de 4D, los segmentos de datos ya no son soportados (el tamaño del archivo de datos es ahora ilimitado). Ahora este comando devuelve sistemáticamente un array con un elemento que contiene el nombre de ruta del archivo de datos de la base.

Entorno del sistema

- Count screens
- Current client authentication
- Current machine
- Current system user
- FONT LIST
- FONT STYLE LIST
- GET SYSTEM FORMAT
- Get system info New 17.0
- Is macOS New 17.0
- Is Windows New 17.0
- LOG EVENT
- Menu bar height
- Menu bar screen
- OPEN COLOR PICKER
- OPEN FONT PICKER
- SCREEN COORDINATES
- SCREEN DEPTH
- Screen height
- Screen width
- Select RGB Color
- SET RECENT FONTS
- SET SCREEN DEPTH
- System folder
- Temporary folder
- *_o_Font name*
- *_o_Font number*
- *_o_Gestalt*
- *_o_PLATFORM PROPERTIES*

Count screens

Count screens -> Resultado

Parámetro	Tipo	Descripción
Resultado	Entero largo	Número de monitores

Descripción

El comando **Count screens** devuelve el número de pantallas conectadas a su equipo.

Current client authentication

Current client authentication {(dominio ; protocolo)} -> Resultado

Parámetro	Tipo		Descripción
dominio	Texto	⇐	Nombre del dominio
protocolo	Texto	⇐	"Kerberos", "NTLM", o cadena vacía
Resultado	Texto	⇒	Nombre de usuario de sesión devuelto por Windows

Descripción

El comando **Current client authentication** pide al servidor Active Directory de Windows autenticar al cliente actual y si tiene éxito, devuelve el nombre de inicio de sesión de Windows para este cliente (identificador de sesión). Si la autenticación falla, se devuelve una cadena vacía.

Este comando sólo se puede utilizar en el contexto de una implementación SSO en Windows con 4D Server. Para más información, por favor consulte la sección **Autenticación única (Single Sign On - SSO) en Windows**.

Por lo general, el cliente y el servidor deben ser gestionados por el mismo Active Directory. Sin embargo, diferentes configuraciones pueden ser soportadas, como se describe en la sección **Configuraciones soportadas**.

La cadena devuelta por el comando debe pasarse a su módulo de identificación 4D para conceder los derechos de acceso al cliente en función de su identificador de sesión de Windows; si define un "usuario por defecto", puede implementar una interfaz en donde el usuario no tenga que volver a introducir los ID (ver ejemplo).

Opcionalmente, el comando puede devolver dos parámetros de texto:

- *dominio*: nombre del dominio al que pertenece el cliente.
- *protocolo*: nombre del protocolo utilizado por Windows para autenticar al usuario. Puede ser "Kerberos" o "NTLM", dependiendo de los recursos disponibles. Si la autenticación falla, se devuelve una cadena vacía ("").

Estos parámetros se pueden utilizar para aceptar o rechazar las conexiones si desea filtrar el acceso en relación con el dominio o el protocolo.

Nivel de seguridad de la autenticación

El nivel de seguridad de la autenticación (es decir, cuánto se puede confiar en el inicio de sesión de usuario) depende de la forma en que el usuario se ha identificado. El valor devuelto en los parámetros del comando **Current client authentication** permitirán conocer en qué se basa el inicio de sesión (si lo hay) y, por tanto, el nivel de seguridad:

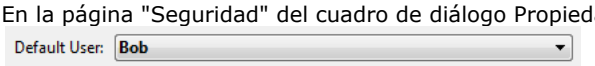
Login	dominio	protocolo	Comentarios
vacío	vacío	vacío	El comando no pudo obtener información de autenticación sobre el usuario que ha iniciado sesión
lleno	vacío	NTLM	El ID devuelto es la local, el cual ha sido definido en el equipo local
lleno	lleno	NTLM	El ID devuelto ha sido autenticado utilizando el protocolo NTLM en el dominio devuelto en el parámetro *dominio*. En este caso, debe seleccionar el dominio para aumentar el nivel de seguridad. Dado que algunas arquitecturas tienen un dominio forest, es necesario asegurarse de que el dominio en el que se ha autenticado el usuario sea el esperado.
lleno	lleno	Kerberos	El ID devuelto ha sido autenticado con el protocolo Kerberos en el dominio esperado. Esta configuración ofrece el más alto nivel de seguridad.

Para más información sobre los requerimientos, por favor consulte el párrafo .

Ejemplo

Usted quiere configurar su aplicación para que los usuarios remotos 4D en Windows se conecten directamente a 4D Server (no se muestra ningún cuadro de diálogo de contraseña), estando conectado con sus derechos actuales. Para ello, es necesario:

1. Habilitar el control de acceso de su base mediante la adición de una contraseña para el Diseñador.
2. En la página "Seguridad" del cuadro de diálogo Propiedades de la base, designar a un usuario como el "usuario por defecto":

   Con esta configuración, no se muestra un diálogo de contraseña para un 4D remoto que se conecta al servidor, todos los clientes se registran como "Bob".
3. En el método base On Server Open Connection, agregue el siguiente código para comprobar la autenticación de usuario del directorio Active:

```
 //Método base On Server Open Connection
C_LONGINT($0;$1;$2;$3)
$login:=Current client authentication($domain;$protocol)
If($login #"")  //un login fue devuelto
   CHANGE CURRENT USER($login;"")  //define el usuario actual
   $0:=0  //acceso aceptado
Else
   $0:=-1  //rechazar la conexión
End if
```

Nota: este escenario básico requiere que los nombres de usuario 4D repliquen los nombres del directorio Active, para un mapeo automático. En una aplicación más sofisticada, la información devuelta por el comando se asigna a una tabla [usuarios] para

crear o seleccionar los usuarios con base en la información del directorio Active.

Current machine

Current machine -> Resultado

Parámetro	Tipo	Descripción
Resultado	Cadena	Nombre del equipo en la red

Descripción

El comando **Current machine** devuelve el nombre de su equipo, como está definido en el panel de control de red.

Ejemplo

Incluso si no está corriendo la versión cliente/servidor de 4D, su aplicación puede incluir servicios de red que necesiten que su equipo esté configurado correctamente. En el **Método de base On Startup** de su aplicación, puede escribir:

```
If((Current machine="")|(Current machine owner=""))
   ` Mostrar una caja de diálogo pidiendo al usuario configurar sus parámetros de red
End if
```

Current system user

Current system user -> Resultado

Parámetro	Tipo	Descripción
Resultado	Cadena	Nombre del dueño del equipo en la red

Descripción

El comando **Current machine owner** devuelve el nombre del dueño de su equipo, tal como está definido en los parámetros de red del sistema operativo.

Ejemplo

Ver el ejemplo del comando **Current machine**.

FONT LIST

FONT LIST (fuentes {; tipoLista})

Parámetro	Tipo		Descripción
fuentes	Array texto	⇐	Array de nombres de fuentes disponibles
tipoLista	Entero largo, Operador	⇒	Tipo de lista de fuente a devolver o * para devolver los nombres de fuente en OS X

Descripción

El comando **FONT LIST** llena el array Texto *fuentes* con los nombres de las fuentes vectoriales disponibles en su sistema.

El parámetro *tipoLista* permite designar el tipo de la lista de fuente a obtener. Para hacerlo, puede pasar una de las siguientes constantes en el parámetro *tipoLista*, disponible en el tema "**Tipo de lista de las fuentes**":

Constante	Tipo	Valor	Comentario
Favorite fonts	Entero largo	1	*fuentes* contiene la lista de fuentes favoritas. - Bajo de Windows: lista de nombres de familias de fuentes activas en el panel de control de Windows. - Bajo OS X: lista de nombres de familias de fuentes de la colección "com.apple.Favorites" que se encuentra en el panel de control, llamada "Favorites" en Inglés, "Favoris" en francés, "Favoriten" en alemán, etc. Esta colección puede estar en blanco si el usuario no ha añadido fuentes favoritas.
Recent fonts	Entero largo	2	*fuentes* contiene la lista de fuentes recientes (lista de fuentes utilizadas durante la sesión 4D). Esta lista es utilizada particularmente por las áreas de texto multiestilo.
System fonts	Entero largo	0	*fuentes* contiene la lista de todas las fuentes del sistema. Opción por defecto si se omite *tipoLista*.

Bajo Mac OS X, cuando se pasa el parámetro opcional *, el comando llena el array *fuentes* con los nombres de las fuentes y no con los nombres de las familias de fuentes. La operación por defecto simplifica la gestión programada de áreas de texto enriquecidas, que utilizan familias de fuente. Si pasa el parámetro *, los nombres de fuente, por ejemplo "Arial bold", "Arial italic", "Arial narrow italic," son devueltos en lugar de las familias, tales como "Arial", "Arial black" o "Arial narrow".

Bajo Windows, el parámetro * no tiene efecto. El comando devuelve siempre las familias de fuentes.

Nota: bajo Mac OS, si utiliza el resultado de este comando con el comando **ST SET ATTRIBUTES** en un área de texto multiestilo, no debe pasar el parámetro * (sólo familias de fuentes soportadas como Attribute font name). Esta limitación no aplica a áreas 4D Write Pro, que aceptan fuentes o nombres de familias de fuentes.

Fuentes vectoriales

Este comando devuelve sólo las fuentes escalables. No se recomienda el uso de fuentes no vectoriales (es decir, fuentes de mapa de bits) para el diseño de interfaces, ya que se basan en una tecnología obsoleta y sufren de limitaciones en cuanto a las variaciones de tamaño. No son compatibles con las funcionalidades más recientes de 4D como las áreas 4D Write Pro.

En OS X, este principio aplica desde OS X 10.4 (las fuentes de mapa de bits QuickDraw son obsoletas a partir de esta versión).

Bajo Windows, este principio se aplica comenzando con 4D v15 R4. Con el fin de ayudar a los desarrolladores a seleccionar sólo fuentes modernas para sus interfaces, sólo las fuentes vectoriales "TrueType" u "OpenType" se listan. Por ejemplo, "ASI_Mono", "MS Sans Serif" y "Syistem" ya no están disponibles. Además, también se ignoran los nombres GDI; sólo los nombres de familias de fuente DirectWrite son soportados. Por ejemplo, las fuentes "Arial Black" o "Segoe UI Black" no están en la lista; Sólo "Arial" y "Segoe" se devuelven.

Notas de compatibilidad para Windows:

- Las fuentes de mapa de bits se pueden seguir utilizando en sus formularios 4D (excepto en las áreas 4D Write Pro). Simplemente se eliminan de la lista devuelta por este comando. Sin embargo, para asegurar la compatibilidad con futuras versiones de 4D y Windows, se recomienda utilizar sólo las familias de fuentes DirectWrite.
- Dado que las fuentes de mapa de bits se filtran desde el parámetro *fuentes* en Windows, la lista resultante es diferente en aplicaciones 4D v15 R4 y superiores, en comparación con versiones anteriores. Por favor asegúrese de adaptar su código si utiliza este comando para seleccionar un tipo de letra no vectorial.

Ejemplo 1

En un formulario, usted quiere obtener una lista desplegable que muestre las fuentes disponibles en el sistema. El método de la lista desplegable es el siguiente:

```
Case of
   :(Form event=On Load)
      ARRAY TEXT(asFuente;0)
      FONT LIST(asFuente)
   ` ...

End case
```

Ejemplo 2

Usted quiere obtener una lista de fuentes recientes:

```
FONT LIST($arrFonts;Recent fonts)
```

FONT STYLE LIST

FONT STYLE LIST (familiaFuiente ; listaEstilosFuente ; listaNomsFuente)

Parámetro	Tipo		Descripción
familiaFuiente	Texto	➡	Nombre de la familia de fuente
listaEstilosFuente	Array texto	⬅	Lista de estilos fuente soportados por la familia de fuente
listaNomsFuente	Array texto	⬅	Lista de nombres completos soportados por la familia de fuente

Descripción

El comando **FONT STYLE LIST** devuelve la lista de estilos y la lista de nombres completos soportados por la familia de fuente designada por el parámetro *familiaFuente*. Este comando le permite diseñar interfaces de manejo de fuentes y estilos, en particular en el contexto de las áreas 4D Write Pro (ver **Referencia 4D Write Pro**).

En *familiaFuente*, pase el nombre de la familia de fuente para el que desea conocer los estilos y nombres de fuentes soportados.

En *listaEstilosFuente*, pase un array texto para ser llenado con la lista de estilos de fuente soportados por la *familiaFuente*. Los estilos se devuelven utilizando sus nombres localizados (es decir, un elemento "cursiva" será devuelto como "Itálico" en un sistema español), por lo que se puede construir un menú pop-up "Estilos" localizado, por ejemplo.

En *listaNomsFuente*, pase un array texto para ser llenado con la lista completa de nombres de fuentes soportadas por la *familiaFuente*. A diferencia del array *listaEstilosFuente*, el array *listaNomsFuente* devuelve los valores no localizados, es decir, los nombres de fuentes basados en la identificación del sistema. Por lo tanto, los nombres de fuentes serán independientes del idioma del sistema. Los elementos de este array son cadenas destinadas a ser utilizadas con el atributo wk font del comando 4D Write Pro **WP SET ATTRIBUTES**. Al utilizar esta funcionalidad, los documentos 4D Write Pro pueden almacenar nombres de fuente y que luego se abrirán en máquinas utilizando cualquier lenguaje del sistema sin problemas de fuentes.

Si la *familiaFuente* no se encuentra en la máquina, los arrays se devuelven vacíos. Para obtener la lista de familias de fuentes disponibles en la máquina, utilice el comando **FONT LIST**.

Ejemplo

Usted desea seleccionar estilos de la familia de fuentes "Verdana" (si está disponible):

```
ARRAY TEXT($aTfonts;0)
ARRAY TEXT($aTStyles;0)
ARRAY TEXT($aTnames;0)
C_LONGINT($numStyle)

FONT LIST($aTfonts)
$numStyle:=Find in array($aTfonts;"Verdana")
If($numStyle#0)
   FONT STYLE LIST($aTfonts{$numStyle};$aTStyles;$aTnames)
End if

 //Por ejemplo, los arrays resultantes son:
 //$aTStyles{1}="Normal"
 //$aTStyles{1}="Italic"
 //$aTStyles{1}="Bold"
 //$aTStyles{1}="Bold Italic"

 // $aTnames{1}="Verdana"
 // $aTnames{1}="Verdana Italic"
 // $aTnames{1}="Verdana Bold"
 // $aTnames{1}="Verdana Bold Italic"
```

GET SYSTEM FORMAT

GET SYSTEM FORMAT (formato ; valor)

Parámetro	Tipo		Descripción
formato	Entero largo	➡	Formato de sistema a recuperar
valor	Cadena	⬅	Formato de sistema a recuperar

Descripción

El comando **GET SYSTEM FORMAT** devuelve el valor actual de varios parámetros regionales definidos en el sistema operativo. Este comando puede utilizarse para crear formatos personalizados "automáticos" basados en las preferencias del sistema.

En el parámetro *formato*, pase el tipo del parámetro del que quiere conocer el valor. El resultado es devuelto directamente por el sistema en el parámetro *valor* como una cadena de caracteres. En *formato*, debe pasar una de las siguientes constantes del tema "". Esta es la descripción de estas constantes:

Constante	Tipo	Valor	Comentario
Currency symbol	Entero largo	2	Símbolo de moneda (ej.: "$")
Date separator	Entero largo	13	Separador utilizado en formatos de fecha (ej.: "/")
Decimal separator	Entero largo	0	Separador decimal (ej.: ".")
Short date day position	Entero largo	15	Posición de día en el formato de fecha corto: "1" = izquierda, "2" = en el medio, "3" = a la derecha
Short date month position	Entero largo	16	Posición del mes en formato de fecha corto: "1" = izquierda, "2" = en el medio, "3" = a la derecha
Short date year position	Entero largo	17	Posición del año en el formato de fecha corto: "1" = izquierda, "2" = medio, "3" = derecha
System date long pattern	Entero largo	8	Formato de salida de fecha largo "dddd MMMM yyyy"
System date medium pattern	Entero largo	7	Formato de salida de fecha medio en la forma "dddd MMMM yyyy"
System date short pattern	Entero largo	6	Formato de salida de fecha corto en la forma "dddd MMMM yyyy"
System time AM label	Entero largo	18	Etiqueta adicional para una hora antes del medio día en formatos de 12 horas (ej.: "Mañana")
System time long pattern	Entero largo	5	Formato de salida de hora largo en la forma "HH:MM:SS"
System time medium pattern	Entero largo	4	Formato de salida de hora medio en la forma "HH:MM:SS"
System time PM label	Entero largo	19	Etiqueta adicional para una hora luego del medio día en formatos de 12 horas (ej.: "tarde")
System time short pattern	Entero largo	3	Formato de salida de hora corto en forma "HH:MM:SS"
Thousand separator	Entero largo	1	Separador de miles (ej.: ",")
Time separator	Entero largo	14	Separador utilizado en formatos de hora (ej.: ":")

Ejemplo

En un cheque que se llena mecánicamente, las sumas escritas por lo general están precedidas de "*" con el fin de evitar fraudes. Si el formato de salida del sistema estándar para la moneda es "$ 5,422.33", el formato de los cheques debe ser del tipo "$***5432.33": sin coma después del miles y sin espacio entre el símbolo $ y el primer número. El formato a utilizar con la función **String** debe ser "$*******.**". Para construirlo por programación es necesario conocer el símbolo monetario y el separador decimal:

```
GET SYSTEM FORMAT(Currency symbol;$vActSim)
GET SYSTEM FORMAT(Decimal separator;$vDecSep)
$MiFormato:="###"+$vActSim+"*******"+$vDecSep+"**"
$Result:=String(cantidad;$MiFormato)
```

Get system info

Get system info -> Resultado

Parámetro	Tipo	Descripción
Resultado	Objeto	Información del sistema

Descripción

El comando **Get system info** devuelve un objeto que contiene información sobre el sistema operativo y las características del hardware y software del sistema de la máquina en que se ejecuta.

El comando devuelve la siguiente información:

Propiedad	Subpropiedad		Tipo	Descripción	Ejemplo
accountName			cadena	El nombre de la cuenta para el usuario actual. Normalmente se usa para identificar una cuenta en el directorio.	"msmith"
cores			número	Número total de núcleos. En el caso de máquinas virtuales, la cantidad total de núcleos asignados.	4
cpuThreads			número	Número total de hilos.	8
machineName			cadena	El nombre de la máquina como se define en los parámetros de red del sistema operativo.	"LAPTOP-M3BLHGSG"
model			cadena	Nombre del modelo del ordenador.	"iMac12,2", "Dell", "Acer", "VMware", etc.
networkInterfaces			colección	Direcciones de red físicas y activas únicamente	
	ipAddresses		colección		
		ip	cadena	La dirección de la interfaz de red	"129.186.81.80"
		type	cadena	El tipo de la interfaz de red	"ipv4", "ipv6"
	name		cadena	El nombre de la interfaz.	"Intel(R) 82574L Gigabit Network Connection"
	type		cadena	El tipo de interfaz (note que el tipo "ethernet" se ofrece para interfaces bluetooth).	"wifi", "ethernet"
osVersion			cadena	La versión del sistema operativo y el número de compilación (*).	"Microsoft Windows 10 Professionnel 10.0.14393"
osLanguage			cadena	Idioma establecido por el usuario actual del sistema. Expresado en el estándar definido por el RFC 3066. Ver **Códigos del lenguaje** en el manual de Diseño para obtener una lista completa.	"fr", "en", "ja", "de", etc.
physicalMemory			número	El volumen de almacenamiento de memoria (en kilobytes) disponible en la máquina.	16777216
processor			cadena	El nombre, tipo y velocidad del procesador.	"Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz"
uptime			número	El tiempo total (en segundos) desde que se inició la máquina.	3600
userName			cadena	El usuario actual en la máquina. Normalmente se utiliza como el nombre a mostrar (es decir, al iniciar sesión en su ordenador).	"Mary Smith"
volumes			collection		
	available		número	El espacio restante que se puede usar.	524288
	capacity		número	Volumen total posible (en kilobytes).	1048576
	disk		objeto | colección (Mac únicamente)		
		description	cadena	Un breve resumen que describe el disco.	"HP LOGICAL VOLUME SCSI Disk Device"
		identifier	cadena	ID de disco(s) (UUID en Mac y disco físico en Windows)	Mac - "87547BDD-EA75-4F48-8BFA-9A7E393EEAB0", Windows - "\\\\.\\PHYSICALDRIVE0"
		size	número	La capacidad total (en kilobytes) del disco	104857600
		interface	cadena	El tipo de interfaz en la máquina.	"USB", "network", "SATA", "SCSI", "cd/dvd", "PCI", etc.
	fileSystem		cadena	El sistema de archivos utilizado por el sistema operativo para almacenar y recuperar archivos en el disco duro.	"NTFS", "Journaled HFS+", "GPFS", etc.
	mountPoint		cadena	El directorio en el sistema de archivos actualmente accesible en el que está montado un sistema de archivos adicional (es decir, conectado lógicamente).	Mac - "/Volumes/Free HD", Windows - "C:"

		Tenga en cuenta que esto está en formato POSIX para Macs.	
name	cadena	solo en Mac - nombre del volumen	"iMac-27-Program6"

(*) Para determinar solo la plataforma que se utiliza, hay dos comandos disponibles: **Is macOS** y **Is Windows**.

Nota: en el caso de las máquinas virtuales, la información devuelta será la de la máquina virtual.

Ejemplo

El siguiente código en una máquina Windows:

```
C_OBJECT($systemInfo)
$systemInfo:=Get system info
```

devuelve un objeto que contiene la siguiente información:


```
{ "title": "Get system info", "machineName":"LAPTOP-M3BLHGSG", "osVersion":"Microsoft Windows 10 Professionnel 10.0.14393",
"osLanguage":"fr", "accountName":"msmith", "userName":"mary smith", "processor":"Intel(R) Core(TM) i7-2600 CPU @ 3.40GH 3.39GHz",
"cores":4, "cpuThreads":8, "networkInterfaces": [ {"type":"ethernet", "name":"Intel(R) 82574L Gigabit Network
Connection", "ipAddresses": [ {"type":"ipV4", "ip":"129.138.10.17"},
{"type":"ipV6","ip":"z1009:0yxw:0000:85v6:0000:0000:ut1s:8001"} ] }, {"type":"wifi", "name":"Wi-Fi",
"ipAddresses": [ {"type":"ipV4", "ip":"129.138.50.8"},
{"type":"ipV6","ip":"a1002:0bc8:0000:85d6:0000:0000:ef1g:7001"} ] } ], "uptime":3600,
"model":"HP", "physicalMemory":16777216, "volumes": [ { "mountPoint": "C:", "capacity": 1048576,
"available": 524288, "fileSystem": "NTFS", "disk": { "identifier": "\\\\.\\PHYSICALDRIVE0",
"interface": "SCSI", "size": 157284382, "description": "Lecteur de disque" }
}, { "mountPoint": "E:", "capacity": 51198972, "available": 51025280, "fileSystem":
"NTFS", "disk": { "identifier": "\\\\.\\PHYSICALDRIVE0", "interface": "SCSI",
"size": 157284382, "description": "Lecteur de disque" } } ] }
```

Is macOS

Is macOS -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		True si el sistema operativo = macOS, de lo contrario False

Descripción

El comando **Is macOS** devuelve True si el sistema operativo actual es macOS.

Ejemplo

Usted desea determinar si el sistema operativo actual es macOS:

```
If(Is macOS)
    ALERT("It's macOS")
Else
    ALERT("It's not macOS")
End if
```

Is Windows

Is Windows -> Resultado

Parámetro	Tipo	Descripción
Resultado	Booleano	True si el sistema operativo = Windows, de lo contrario False

Descripción

El comando **Is Windows** devuelve True si el sistema operativo actual es Windows.

Ejemplo

Desea determinar si el sistema operativo actual es Windows:

```
If(Is Windows)
    ALERT("Es Windows")
Else
    ALERT("No es Windows")
End if
```

LOG EVENT

LOG EVENT ({tipoSalida ;} mensaje {; importancia})

Parámetro	Tipo		Descripción
tipoSalida	Entero largo	➡	Tipo de salida del mensaje
mensaje	Cadena	➡	Contenido del mensaje
importancia	Entero largo	➡	Nivel de importancia del mensaje (sólo para Windows)

Descripción

El comando **LOG EVENT** permite configurar un sistema personalizado de registro de eventos internos que ocurren durante el uso de su aplicación.

Pase en el parámetro *mensaje* la información personalizada a notar en función del evento.

El parámetro opcional *tipoSalida* permite precisar el canal de salida tomado por el *mensaje*. Puede pasar en este parámetro una de las siguientes constantes, ubicadas en el tema **Historial de eventos**:

Constante	Tipo	Valor	Comentario
Into 4D commands log	Entero largo	3	Indica a 4D grabar el *mensaje* en el archivo de historial de los comandos de 4D, si este archivo se ha activado. *El archivo de historial de comandos de 4D puede activarse utilizando el comando **SET DATABASE PARAMETER** (selector 34).* **Nota:** los archivos de historial de 4D se agrupan en la carpeta **Logs**, creada junto al archivo de estructura de la base (ver el comando ***Get 4D folder***).
Into 4D debug message	Entero largo	1	Indica a 4D enviar el *mensaje* al entorno de depuración del sistema. El resultado depende de la plataforma: • Bajo Mac OS: el comando envía el mensaje a la Consola • Bajo Windows: el comando envía el mensaje como un mensaje de depuración. Para poder leer este mensaje, debe tener Microsoft Visual Studio o DebugView para Windows (http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx)
Into 4D diagnostic log	Entero largo	5	Le indica a 4D poner el mensaje en el archivo de diagnóstico de 4D, si este archivo está activo. El archivo de diagnóstico puede activarse con ayuda del comando **SET DABABASE PARAMETER** (*selector 79*).
Into 4D request log	Entero largo	2	Indica a 4D grabar el *mensaje* en el archivo de historial de peticiones de 4D, si este archivo ha sido activado
Into Windows log events	Entero largo	0	Indica a 4D enviar el *mensaje* "Log events" de Windows. Este historial recibe y almacena los mensajes que vienen de las aplicaciones en ejecución. En este caso, puede definir el nivel de importancia del *mensaje* vía el parámetro opcional *importancia* (ver a continuación). **Notas:** • Para que esta funcionalidad esté disponible, el servicio Log Events de Windows debe estar en ejecución. • Bajo Mac OS, el comando no hace nada con este tipo de salida

Si omite el parámetro *tipoSalida,* el valor 0 se utiliza por defecto (Into Windows Log Events).

Si ha definido un *tipoSalida* de tipo Into Windows Log Events, puede atribuir al mensaje un nivel de importancia vía el parámetro opcional *importancia* con el fin de facilitar la lectura del historial de eventos. Hay tres niveles de importancia: Información, Advertencia y Error. 4D ofrece las siguientes constantes predefinidas, ubicadas en el tema **Historial de eventos**:

Constante	Tipo	Valor
Error message	Entero largo	2
Information message	Entero largo	0
Warning message	Entero largo	1

Si no pasa el parámetro *importancia* o si pasa un valor invalido, se utiliza el valor por defecto (0).

Ejemplo

Si quiere realizar un seguimiento de las aperturas de su base bajo Windows, puede escribir la siguiente línea de código en el **Método de base On Startup**:

```
LOG EVENT(Into Windows log events;"Apertura de la base Facturación.")
```

Cada vez que se abre la base, esta información se escribirá en el visor de eventos de Windows y su nivel de importancia será 0.

Menu bar height

Menu bar height -> Resultado

Parámetro	Tipo	Descripción
Resultado	Entero largo	Altura (expresada en píxeles) de la barra de menús (devuelve cero si la barra de menús está oculta)

Descripción

Menu bar height devuelve la altura de la barra de menús, expresada en píxeles.
El comando devuelve 0:

- Si la barra de menús está oculta
- En modo SDI en Windows, si se llama desde un proceso sin una ventana de formulario. Para más información sobre este modo, consulte la sección **Modo SDI en Windows**.

Nota: cuando la aplicación se ejecuta en modo SDI en Windows, **Menu bar height** devuelve la altura de una sola línea de barra de menú, incluso si la ventana se estrecha y la barra de menús se ha ajustado en dos o más líneas.

Menu bar screen

Menu bar screen -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo	⮌	Número de la pantalla que contiene la barra de menús

Descripción

Menu bar screen devuelve el número de la pantalla donde está ubicada la barra de menús.
Nota Windows: en Windows, **Menu bar screen** generalmente devuelve 1.

OPEN COLOR PICKER

OPEN COLOR PICKER {(textOFondo)}

Parámetro	Tipo		Descripción
textOFondo	Entero largo	➡	0 o si se omite = color del texto, 1 = color del fondo del texto

Descripción

El comando **OPEN COLOR PICKER** muestra la caja de diálogo de selección de color del sistema.

Nota: esta caja de diálogo es modal en Windows pero no en OS X.

Si el usuario selecciona un color y valida la caja de diálogo, este color se aplica a la selección actual de texto en el objeto con el foco, si la propiedad "Aceptar selector color/fuente" está seleccionada para este objeto (ver el manual de *Diseño*).

Si pasa 0 en el parámetro *textOFondo* u omite este parámetro, el color seleccionado se aplica al texto. Si pasa 1 en *textOFondo*, este color se aplica al fondo del texto.

Si el color se ha cambiado, el evento formulario On After Edit se genera para el objeto.

OPEN FONT PICKER

OPEN FONT PICKER
Este comando no requiere parámetros

Descripción

El comando **OPEN FONT PICKER** muestra la caja de diálogo de selección de fuente del sistema.

Nota: esta caja de diálogo es modal en Windows pero no en OS X.

Si el usuario selecciona una fuente y/o un estilo y valida la caja de diálogo, los cambios se aplican a la selección actual de texto en el objeto con el foco, si la propiedad "Aceptar selector color/fuente" está seleccionada para este objeto (ver el manual de *Diseño*). De lo contrario, el comando no hace nada.

Si se cambia el tipo de fuente, el evento formulario On After Edit se genera para el objeto.

Ejemplo

En un formulario, usted desea añadir un botón para mostrar el selector de fuente con el fin de permitir a los usuarios modificar la fuente o el estilo de un área de variable texto. Asegúrese de que:

- la variable texto tiene la propiedad "Permitir selector fuente/color" seleccionada.
- la propiedad "Enfocable" para el botón ha sido deseleccionada.

Este es el código del botón:

```
Case of
   :(Form event=On Clicked)
      GOTO OBJECT(textVar)  //da el foco a la variable
      OPEN FONT PICKER
End case
```

SCREEN COORDINATES

SCREEN COORDINATES (izquierda ; superior ; derecha ; inferior {; pantalla})

Parámetro	Tipo		Descripción
izquierda	Entero largo	⇐	Coordenada izquierda del área de pantalla
superior	Entero largo	⇐	Coordenada superior del área de la pantalla
derecha	Entero largo	⇐	Coordenada derecha del área de la pantalla
inferior	Entero largo	⇐	Coordenada inferior del área de la pantalla
pantalla	Entero largo	⇒	Número de la pantalla, o pantalla principal si se omite

Descripción

El comando **SCREEN COORDINATES** devuelve en los parámetros *izquierda*, *arriba*, *derecha,* y *abajo* las coordenadas de la pantalla especificada por *pantalla*.

Si omite el parámetro *pantalla*, el comando devuelve las coordenadas de la pantalla principal.

SCREEN DEPTH

SCREEN DEPTH (profundidad ; color {; pantalla})

Parámetro	Tipo		Descripción
profundidad	Entero largo	⇐	Profundidad de la pantalla (número de colores = 2 ^ profundidad)
color	Entero largo	⇐	1 = Pantalla color 0 = Pantalla blanco y negro o escala de grises
pantalla	Entero largo	⇒	Número de la pantalla, o pantalla principal si se omite

Descripción

El comando **SCREEN DEPTH** devuelve en los parámetros *profundidad* y *color* la información sobre el monitor.
Después de la llamada:

- Se devuelve la profundidad de la pantalla en *profundidad*. La profundidad de la pantalla elevada como potencia de 2 permite conocer el número de colores mostrados en su monitor. Por ejemplo, si su monitor está definido para 256 colores (2^8), la profundidad de su pantalla es 8. Las siguientes son constantes predefinidas ofrecidas por 4D:

Constante	Tipo	Valor
Black and white	Entero largo	0
Four colors	Entero largo	2
Sixteen colors	Entero largo	4
Two fifty six colors	Entero largo	8
Thousands of colors	Entero largo	16
Millions of colors 24 bit	Entero largo	24
Millions of colors 32 bit	Entero largo	32

Si el monitor está configurado para mostrar colores, se devuelve *1* en *color*. Si el monitor está configurado para mostrar escala de grises, se devuelve *0* en *color*. Note que este valor es significativo en la plataforma Macintosh. Las siguientes son constantes predefinidas ofrecidas por 4D:

Constante	Tipo	Valor
Is gray scale	Entero largo	0
Is color	Entero largo	1

- El parámetro opcional *pantalla* especifica el monitor para el cual quiere obtener la información. Si omite el parámetro *pantalla*, el comando devuelve la profundidad de la pantalla principal.

Ejemplo

Su aplicación muestra varios gráficos a color. Puede escribir en alguna parte en su base:

```
SCREEN DEPTH($vlProf;$vlColor)
If($vlProf<8)
   ALERT("Los formularios se verían mejor si el monitor"+" estuviera configurado para mostrar 256 colores o más.")
End if
```

Screen height

Screen height {(*)} -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Windows: altura de la ventana de la aplicación o altura de la pantalla si se especifica * Macintosh: altura de la pantalla principal
Resultado	Entero largo	↩	Altura expresada en píxeles

Descripción

En Windows, **Screen height** devuelve la altura de la ventana de la aplicación 4D (ventana MDI). Si pasa el parámetro opcional *, **Screen height** devuelve la altura de la pantalla.

En Mac OS, **Screen height** devuelve la altura de la pantalla principal, es decir la pantalla donde está ubicada la barra de menús.

Screen width

Screen width {(*)} -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Windows: largo de la ventana de la aplicación o largo de la pantalla si se especifica * Macintosh: largo de la pantalla principal
Resultado	Entero largo	↩	Largo expresado en píxeles

Descripción

En Windows, **Screen width** devuelve el largo de la ventana de la aplicación 4D (ventana MDI). Si pasa el parámetro opcional *, **Screen width** devuelve el largo de la pantalla.

En Macintosh, **Screen width** devuelve el largo de la pantalla principal, es decir el largo de la pantalla que contiene la barra de menús.

Select RGB Color

Select RGB Color {(colorDefecto {; mensaje})} -> Resultado

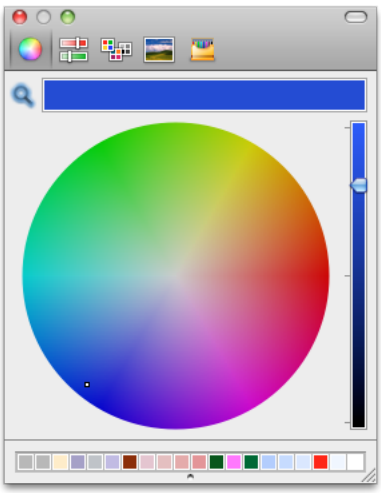
Parámetro	Tipo		Descripción
colorDefecto	Entero largo	➡	Color RGB preseleccionado
mensaje	Alpha	➡	Título de la ventana de selección
Resultado	Entero largo	⮌	Color RGB

Descripción

El comando **Select RGB Color** muestra la ventana sistema de selección de color y devuelve el valor RGB del color seleccionado por el usuario.

La ventana sistema de selección de color se ve de esta manera:

Macintosh



Windows

El parámetro opcional *colorDefecto* permite preseleccionar un color en la ventana. Este parámetro permite por ejemplo restaurar por defecto el último color definido por el usuario. Pase en este parámetro un valor de color formato RGB (para mayor información, consulte la descripción del comando **OBJECT SET RGB COLORS**). Puede utilizar una de las constantes del tema **DEFINIR COLORES RVA**. Si se omite el parámetro *colorDefecto* o si pasa 0, el color negro es seleccionado al abrir la caja de diálogo.

El parámetro opcional *mensaje* permite personalizar el título de la ventana sistema. Por defecto, si se omite este parámetro, aparece el título "Colores".

El efecto de la validación de esta caja de diálogo difiere dependiendo de la plataforma:

- Bajo Windows, si el usuario hace clic en el botón **OK**, el comando devuelve el valor de color seleccionado en formato RGB y la variable sistema *OK* toma el valor 1. Si el usuario cancela la caja de diálogo, el comando devuelve -1 y la variable sistema *OK* toma el valor 0.
- Bajo Mac OS, sólo puede cerrar esta caja de diálogo haciendo clic en la casilla de cierre o presionando la tecla **Esc**. En ambos caos, la variable sistema *OK* toma el valor 1, sin importar las acciones del usuario en la ventana. El comando devuelve el valor del color seleccionado en formato RGB. Si el usuario no selecciona un color, el valor devuelto es el pasado en *colorDefecto* (si lo hay) o 0 si no se pasa *colorDefecto*.

Nota: este comando no debe ejecutarse en el equipo servidor ni en un proceso web.

SET RECENT FONTS

SET RECENT FONTS (arrayFuentes)

Parámetro	Tipo		Descripción
arrayFuentes	Array texto	➡	Array de nombres de fuentes

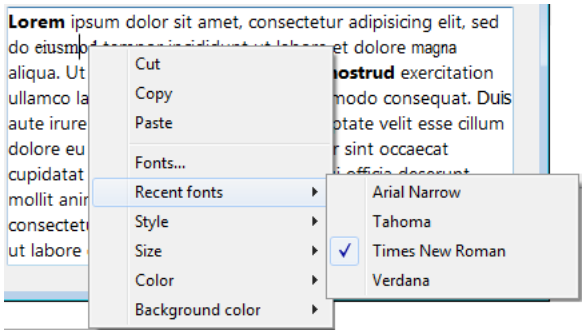
Descripción

El comando **SET RECENT FONTS** modifica la lista de fuentes recientes que aparecen en el menú contextual de las "fuentes recientes" .

Este menú contiene los nombres de las últimas fuentes seleccionadas durante la sesión. Se utiliza, en particular, para áreas **Interacción de comandos genéricos con textos multiestilos**.

Ejemplo

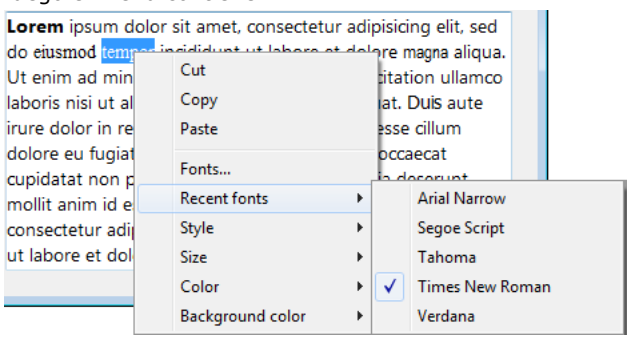
Usted quiere añadir una fuetne al menú de fuentes recientes:



Ejecuta el siguiente código:

```
ARRAY TEXT($arrRecent;0)
FONT LIST($arrRecent;2)
APPEND TO ARRAY($arrRecent;"Segoe Script")
APPEND TO ARRAY($arrRecent)
```

Luego el menú contiene:



SET SCREEN DEPTH

SET SCREEN DEPTH (profundidad {; color {; pantalla}})

Parámetro	Tipo		Descripción
profundidad	Entero largo	➡	Profundidad de la pantalla (número de colores = 2 ^ Profundidad de pantalla)
color	Entero largo	➡	1 = Color, 0 = Escala de grises
pantalla	Entero largo	➡	Número de la pantalla, o pantalla principal, si se omite

Descripción

SET SCREEN DEPTH permite modificar la profundidad y los parámetros de colores/escala de grises de la pantalla cuyo número se pasa en *pantalla*. Si omite este parámetro, el comando se aplica a la pantalla principal.

Para conocer los valores a pasar en *color* y *profundidad*, consulte la descripción del comando **SCREEN DEPTH**.

System folder

System folder {(tipo)} -> Resultado

Parámetro	Tipo		Descripción
tipo	Entero largo	⇒	Tipo de carpeta sistema
Resultado	Cadena	⇐	Ruta de acceso de una carpeta del sistema activo

Descripción

El comando **System folder** devuelve la ruta de acceso a una carpeta particular del sistema operativo o a la carpeta activa del sistema Windows o Mac OS.

Pase un valor en el parámetro opcional *tipo* indicando el tipo de carpeta del sistema. 4D le ofrece las siguientes constantes predefinidas, ubicadas en el tema "**Carpeta sistema**":

Constante	Tipo	Valor	Comentario
Applications or program files	Entero largo	16	
Desktop	Entero largo	15	
Documents folder	Entero largo	17	Carpeta "Documents" del usuario
Favorites Win	Entero largo	14	
Fonts	Entero largo	1	
Start menu Win_all	Entero largo	8	
Start menu Win_user	Entero largo	9	
Startup Win_all	Entero largo	4	
Startup Win_user	Entero largo	5	
System	Entero largo	0	
System Win	Entero largo	12	
System32 Win	Entero largo	13	
User preferences_all	Entero largo	2	
User preferences_user	Entero largo	3	

Notas

- Las constantes con sufijo **Win**, pueden utilizarse bajo Windows únicamente. Cuando se utilizan en Mac OS, **System folder** devuelve una cadena vacía.
- Las rutas de acceso a algunas carpetas sistema pueden especificar el usuario actual. Las constantes 2 a 9 le permiten elegir si quiere obtener la ruta de acceso a una carpeta común para todos los usuarios o la ruta personalizada para el usuario actual.

Si omite el parámetro *tipo*, la función devolverá la ruta a la carpeta sistema activa (= constante System).

Temporary folder

Temporary folder -> Resultado

Parámetro	Tipo		Descripción
Resultado	Cadena		Ruta de acceso a la carpeta temporal

Descripción

El comando **Temporary folder** devuelve la ruta de acceso a la carpeta temporal actual definida por su sistema.

Ejemplo

Ver el ejemplo del comando **APPEND DATA TO PASTEBOARD**.

_o_Font name

_o_Font name (numero) -> Resultado

Parámetro	Tipo		Descripción
numero	Entero largo	⇒	Número de la fuente de la cual recuperar el nombre
Resultado	Cadena	⇐	Nombre de la fuente

Descripción

Este comando es obsoleto y no debe ser utilizado a partir de 4D v14. Se mantiene por razones de compatibilidad pero no será soportado en futuras versiones del programa.

_o_Font number

_o_Font number (nomFuente) -> Resultado

Parámetro	Tipo		Descripción
nomFuente	Cadena	⇒	Nombre de la fuente de la cual devolver el número de fuente
Resultado	Entero largo	⇐	Número de fuente

Descripción

Este comando es obsoleto y no debe utilizarse a partir de 4D v14. Se mantiene por razones de compatibilidad pero no será soportado en futuras versiones del programa.

_o_Gestalt

_o_Gestalt (selector ; valor) -> Resultado

Parámetro	Tipo		Descripción
selector	Cadena	➡	Selector gestalt (4 caracteres)
valor	Entero largo	⬅	Resultado de gestalt
Resultado	Entero largo	↩	Código del error resultante

Nota de compatibilidad

*Este comando está en desuso en 4D v17 y superior. Ahora puede usar el comando **Get system info** para obtener información completa del sistema.*

_o_PLATFORM PROPERTIES

_o_PLATFORM PROPERTIES (plataforma {; sistema {; procesador {; lenguaje}}})

Parámetro	Tipo		Descripción
plataforma	Entero largo	⇐	2 = Mac OS, 3 = Windows
sistema	Entero largo	⇐	Depende de la versión que utilice
procesador	Entero largo	⇐	Familia del procesador
lenguaje	Entero largo	⇐	Depende del sistema que utilice

Nota de compatibilidad

*Este comando es obsoleto en 4D v17 y superiores. Puede utilizar el comando **Get system info** para obtener información completa del sistema, o los comandos **Is macOS**, o **Is Windows** solo para información de la plataforma.*

Entrada de datos

- *ACCEPT*
- *ADD RECORD*
- *CANCEL*
- *DIALOG*
- *Modified*
- *MODIFY RECORD*
- *Old*
- *REJECT*
- *_o_ADD SUBRECORD*
- *_o_MODIFY SUBRECORD*

ACCEPT

ACCEPT
Este comando no requiere parámetros

Descripción

*El comando **ACCEPT** se utiliza en métodos de objeto o de formulario (o en subrutinas) para:*

- *aceptar un registro o subregistro nuevo o modificado, para el cual la entrada de datos ha sido inicializada utilizando **ADD RECORD**, **MODIFY RECORD**, **ADD SUBRECORD**, o **MODIFY SUBRECORD**.*
- *aceptar un formulario mostrado por el comando **DIALOG**.*
- *salir de un formulario que muestra una selección de registros, utilizando **DISPLAY SELECTION** o **MODIFY SELECTION**.*

***ACCEPT** efectúa la misma acción que si un usuario hubiera presionado la tecla Intro. Después de que el formulario es aceptado, la variable sistema OK toma el valor 1.*

***ACCEPT** se ejecuta con frecuencia como resultado de la selección de un comando de menú. **ACCEPT** es igualmente utilizado en el método de objeto de un botón "sin acción".*

*Este comando también puede ser utilizado en el método de caja de cierre opcional de una ventana creada por el comando **Open window**. Si hay una caja de control de menú en una ventana, se puede llamar a **ACCEPT** o **CANCEL**, en el método a ejecutar, cuando se haga doble clic sobre la caja de control de menú o se seleccione el comando de menú **Cerrar**.*

*No es posible encadenar varios **ACCEPT**. La ejecución consecutiva de dos comandos **ACCEPT** en un método tendrá el mismo resultado que la ejecución de un solo comando.*

ADD RECORD

ADD RECORD ({tabla}{;}{*})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a utilizar para entrada de datos o Tabla por defecto, si se omite
*		➡	Ocultar barras de desplazamiento

Compatibilidad

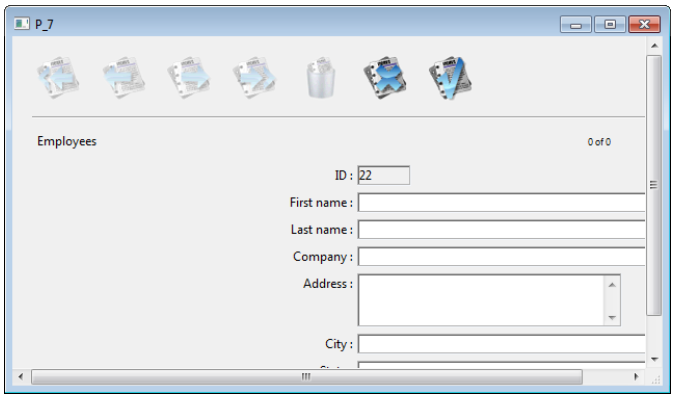
Este comando se implementó en las primeras versiones de 4D y sigue siendo útil para la creación de prototipos o desarrollos básicos. Sin embargo, para construir interfaces personalizadas y modernas, ahora se recomienda utilizar formularios genéricos basados en el comando ***DIALOG*** *que ofrecen funciones avanzadas y un mejor control sobre el flujo de datos.*

Descripción

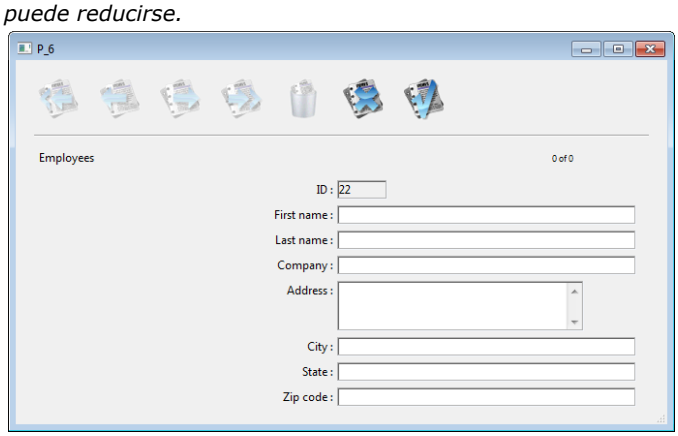
El comando ***ADD RECORD*** *permite al usuario añadir un nuevo registro en la tabla tabla o en la tabla por defecto, si omite el parámetro tabla.*

ADD RECORD *crea un nuevo registro, hace del nuevo registro el registro actual para el proceso actual y muestra el formulario de entrada actual. En el entorno Menús personalizados, después de que el usuario acepta el nuevo registro, el nuevo registro es el único registro en la selección actual.*

La siguiente imagen presenta un formulario de entrada de datos típico.



*El formulario aparece en la ventana del primer plano del proceso. La ventana tiene barras de desplazamiento y una caja de control del tamaño. Si pasa el parámetro opcional * las barras de desplazamiento no aparecen y la ventana del formulario no puede reducirse.*



ADD RECORD *muestra el formulario hasta que el usuario acepta o cancela el registro. Si el usuario está añadiendo varios registros, el comando debe ejecutarse una vez para cada registro.*

El registro se guarda (aceptado) si el usuario hace clic en el botón Aceptar o al presionar la tecla Intro (teclado numérico), o si se ejecuta el comando ***ACCEPT.***

El registro no se guarda (cancelado) si el usuario hace clic en el botón Cancelar o presiona la tecla de anulación Esc o si se ejecuta el comando ***CANCEL****.*

Nota: *este comando no requiere tabla para estar en modo lectura/escritura. Puede ser utilizado incluso si la tabla está en modo lectura únicamente (ver* ***Record Locking****).*

Después de llamar a ***ADD RECORD****, OK toma el valor 1 si se acepta el registro y 0 si se cancela.*

Nota: *el registro permanece en memoria, incluso cuando se cancela, y puede guardarse si se ejecuta* ***SAVE RECORD*** *antes de que cambie el puntero del registro actual.*

Ejemplo 1

El siguiente ejemplo es un bucle utilizado generalmente para añadir nuevos registros a una base:

```
FORM SET INPUT([Clientes];"Entrada") ` Designar el formulario de entrada de la tabla [Clientes]
Repeat ` Bucle hasta que el usuario cancele
   ADD RECORD([Clientes];*) ` Añadir un registro a la tabla [Clientes]
Until(OK=0) ` Hasta que el usuario cancele
```

Ejemplo 2

El siguiente ejemplo busca un cliente en la base. Dependiendo de los resultado de la búsqueda, sucederá una de estas cosas. Si no se encuentra un cliente, entonces se le permite al usuario añadir un nuevo cliente con ***ADD RECORD****. Si se encuentra al menos un cliente, se le presenta al usuario el primer registro encontrado, el cual puede modificarse con* ***MODIFY RECORD****:*

```
READ WRITE([Clientes])
FORM SET INPUT([Clientes];"Entrada") ` Designar el formulario de entrada
vlCustNum:=Num(Request("Introducir un número de cliente:")) ` Obtener el número de cliente
If(OK=1)
   QUERY([Clientes];[Clientes]CustNo=vlCustNum) ` Buscar el cliente
   If(Records in selection([Clientes])=0) ` si no se encuentra ningún cliente···
      ADD RECORD([Clientes]) ` Añadir un nuevo registro
   Else
      If(Not(Locked([Clientes])))
         MODIFY RECORD([Clientes]) ` Modificar el registro
         UNLOAD RECORD([Clientes])
      Else
         ALERT("El registro está siendo utilizado actualmente.")
      End if
   End if
End if
```

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si se acepta el registro y 0 si se cancela. La variable OK no toma ningún valor hasta que el registro haya sido validado o anulado.

CANCEL

CANCEL
Este comando no requiere parámetros

Descripción

*El comando **CANCEL** se utiliza en métodos de objeto o de formulario (o en una subrutina) para:*

- *cancelar la creación o modificación de un registro o subregistro, para el cual la entrada de datos ha sido inicializada utilizando **ADD RECORD**, **MODIFY RECORD**, **ADD SUBRECORD** o **MODIFY SUBRECORD**.*
- *cancelar un formulario mostrado por intermedio del comando **DIALOG**.*
- *salir de un formulario que muestra una selección de registros, utilizando **DISPLAY SELECTION** o **MODIFY SELECTION**.*
- *cancelar la impresión de un formulario que está a punto de ser impreso utilizando el comando **Print form** (ver a continuación).*

*En el contexto de entrada de datos, **CANCEL** efectúa la misma acción que si el usuario hubiera presionado la tecla de cancelación (**Esc**).*

__CANCEL__ se ejecuta con frecuencia como resultado de la selección de un comando de menú. __CANCEL__ también se utiliza con frecuencia en el método de objeto de un botón "sin acción".

*Este comando también se usa en el metódo de la caja de cierre opcional de una ventana creada por el comando **Open window**. Si hay una ventana con caja de control de menú, se puede llamar a **ACCEPT** o **CANCEL**, en el método a ejecutar, cuando se haga doble clic en en la caja de control de menú o se seleccione el comando de menú **Cerrar**.*

*No es posible encadenar varios **CANCEL**. La ejecución consecutiva de dos comandos **CANCEL** en un método tendrá el mismo resultado que la ejecución de un solo comando.*

*Finalmente, este comando puede ser utilizado en el evento de formulario **On Printing Detail**, cuando se utiliza el comando **Print form**. En este contexto, el comando **CANCEL** suspende la impresión del formulario que está a punto de imprimirse, luego retoma en la siguiente página. Este mecanismo puede utilizarse para administrar la impresión de formularios cuando no hay suficiente espacio o en caso de que sea necesaria una ruptura de página.*

***Nota:** esta operación es diferente de la del comando **PAGE BREAK(*)** que cancela TODOS los formularios que están en espera de impresión.*

Ejemplo

*Consulte el ejemplo del comando **SET PRINT MARKER**.*

Variables y conjuntos del sistema

*Cuando el comando **CANCEL** se ejecuta (anulación de formulario o de impresión), la variable sistema OK toma el valor 0.*

DIALOG

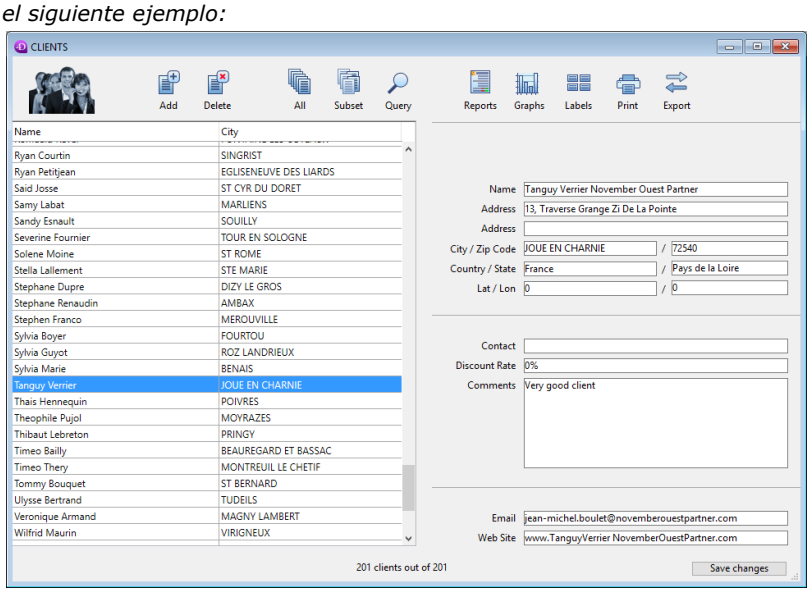
DIALOG ({tabla ;} form {; formData}{; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a la cual pertenece el formulario o Si se omite: tabla por defecto o uso del formulario de proyecto
form	Cadena, Objeto	➡	Nombre de tabla o formulario de proyecto a mostrar como diálogo
formData	Objeto	➡	Datos para asociar al formulario
*	Operador	➡	Utilizar el mismo proceso

Descripción

*El comando **DIALOG** presenta el formulario al usuario junto con los parámetros formData (opcional). Este comando se designa para trabajar con interfaces de usuario avanzadas y personalizadas basadas en formularios. Puede utilizarlo para mostrar información de la base o de otras ubicaciones, o para ofrecer funcionalidades de entrada de datos. A diferencia de **ADD RECORD** o **MODIFY RECORD**, **DIALOG** le da control total sobre el formulario, su contenido y los bontones de navegación y validación.*

*Este comando por lo general se llama junto con **Open form window** para mostrar formularios sofisticados, como se muestra en el siguiente ejemplo:*



*Utilice **DIALOG** en lugar de **ALERT**, **CONFIRM** o **Request** cuando la información qie sea presentada o reunida sea más compleja de lo que estos comandos pueden manejar.*

***Nota:** en bases convertidas, es posible prohibir la entrada de datos en cajas de diálogo (y limitar la entrada de datos a variables únicamente) utilizando una opción en las Preferencias de 4D (página Compatibilidad). Esta restricción corresponde al funcionamiento de versiones anteriores de 4D.*

En el parámetro formulario, puede pasar:

- *el nombre de un formulario (formulario proyecto o formulario tabla) a utilizar;*
- *la ruta (en sintaxis POSIX) a un archivo .json válido que contiene una descripción del formulario a utilizar. Ver **Ruta de archivo del formulario**;*
- *un objeto que contiene una descripción del formulario a utilizar.*

*Opcionalmente, puede pasar parámetros al form mediante el objeto formData. Toda propiedad del objeto formData estará disponible desde el contexto del formulario mediante el comando **Form**. Por ejemplo, si pasa un objeto que contiene {"version","12"} en formData, podrá obtener el valor de la propiedad "Version" en el formulario llamando:*

```
$v:=Form.version //"12"
```

*Utilizando una variable local para formData, esta funcionalidad le permite pasar parámetros de manera segura a sus formularios, cualquiera que sea el contexto de la llamada. En particular, si el mismo formulario se llama desde diferentes lugares en el mismo proceso, siempre podrá acceder a sus valores específicos simplemente llamando a **Form**.myProperty. Además, dado que los objetos se pasan por referencia, si el usuario modifica un valor de propiedad en el formulario, éste se guardará automáticamente en el mismo objeto.*
*Al combinar el objeto formData y el comando **Form**, puede enviar parámetros al formulario o leer los parámetros en cualquier momento con código limpio y seguro.*

***Nota:** si no pasa el parámetro formData o si pasa un objeto indefinido, **DIALOG** crea automáticamente un nuevo objeto vacío vinculado al form y disponible a través del comando **Form**.*

*El diálogo es cerrado por el usuario con una acción Aceptar (disparada por la acción estándar ak accept, la tecla Intro o el comando **ACCEPT**). Una acción aceptar definirá la variable sistema OK en 1, mientra una acción cancelar definirá OK en 0. Recuerde que validar no es igual a guardar: si el diálogo incluye campos, debe llamar explícitamente al comando **SAVE RECORD** para guardar los datos que hayan sido modificados.*
*El diálogo se cancela si el usuario hace clic en el botón Cancelar (**Escape** en Windows, **Esc** en Macintosh) o si se ejecuta el comando **CANCEL**.*

*Si pasa el parámetro opcional *, el formulario se carga y muestra en la última ventana del proceso actual y el comando termina su ejecución mientras deja el formulario activo en pantalla. Este formulario luego reacciona "normalmente" a las acciones del*

*usuario y se cierra vía una acción estándar o cuando el código 4D relacionado con el formulario (método objeto o método formulario) llama al comando **CANCEL** o **ACCEPT**. Si el proceso actual termina, los formularios creados de esta forma se cierran automáticamente de la misma forma que si se hubiera llamado el comando **CANCEL**. Este modo de apertura es particularmente útil para mostrar una paleta flotante con un documento, sin necesidad de otro proceso.*

Notas:

- *Puede combinar el uso de la sintaxis **DIALOG**(form;*) con el comando **CALL FORM** para establecer la comunicación entre los formularios.*
- *debe crear una ventana antes de llamar la instrucción **DIALOG**(form;*); si no es posible utilizar la ventana de diálogo actual en el proceso ni la ventana creada por defecto para cada proceso. De lo contrario, se genera el error -9909.*
- *cuando se utiliza el parámetro *, la ventana se cierra automáticamente después de una acción estándar o una llamada al comando **CANCEL** o **ACCEPT**. No tiene que gestionar el cierre de la propia ventana.*

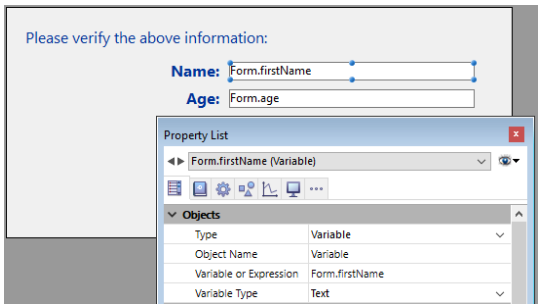
Ejemplo 1

El siguiente ejemplo puede usarse para crear una paleta de herramientas

```
 `Muestra la paleta de herramientas
$palette_window:=Open form window("tools";Palette form window)
DIALOG("tools";*) `Give back the control immediately
 `Muestra ventana del documento principal
$document_window:=Open form window("doc";Standard form window)
DIALOG("doc")
```

Ejemplo 2

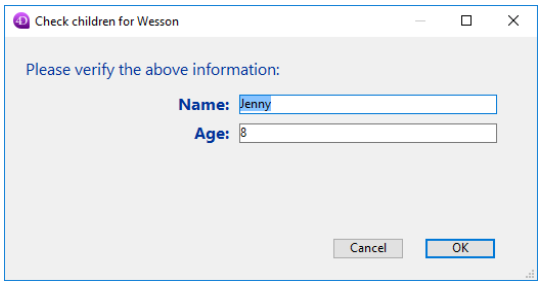
*En un formulario, usted asignó algunas propiedades de objeto **Form** a variables:*



Luego, puede ejecutarlas desde cualquier lugar de la aplicación:

```
C_LONGINT($win)
$win:=Open form window("Edit_Address";Movable form dialog box;Horizontally centered;Vertically centered)
DIALOG("Edit_Address";New object("firstName";"Mike";"age";12))
CLOSE WINDOW($win)
```

El formulario muestra los valores que ha pasado:



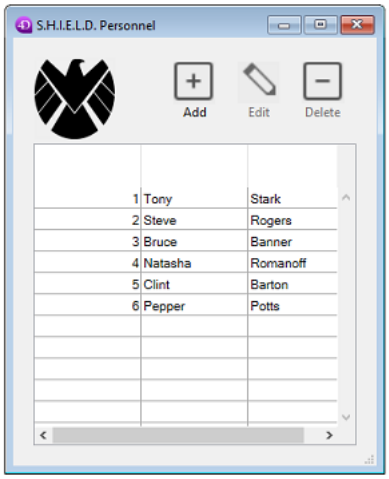
***Nota:** este ejemplo requiere que la notación de objeto esté habilitada en la base de datos (ver **Página Compatibilidad**).*

Ejemplo 3

El siguiente ejemplo usa la ruta a un formulario .json para mostrar los registros en una lista de empleados:

```
Open form window("/RESOURCES/OutputPersonnel.json";Plain form window)
ALL RECORDS([Personnel])
DIALOG("/RESOURCES/OutputPersonnel.json";*)
```

que devuelve:

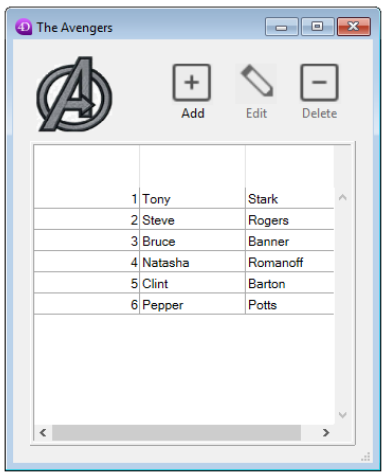


Ejemplo 4

El siguiente ejemplo utiliza un archivo .json como un objeto y modifica algunas propiedades:

```
C_OBJECT($form)
$form:=JSON Parse(Document to text(Get 4D folder(Current resources folder)+"OutputPersonnel.json"))
$form.windowTitle:="The Avengers"
$form.pages[1].objects.logo.picture:="/RESOURCES/Images/Avengers.png"
$form.pages[1].objects.myListBox.borderStyle:="double"
Open form window($form;Plain form window)
DIALOG($form;*)
```

El formulario modificado se devuelve con el título, el logotipo y el borde modificados:



Variables y conjuntos del sistema

*Después de llamar a **DIALOG**, si se acepta el diálogo, la variable sistema OK toma el valor 1; si se cancela toma el valor 0.*

Modified

Modified (campo) -> Resultado

Parámetro	Tipo		Descripción
campo	Campo	⇒	Campo a probar
Resultado	Booleano	⇐	Verdadero si se ha asignado un nuevo valor al campo, de lo contrario Falso

Descripción

Modified devuelve **True** si un valor ha sido asignado por programación a campo o si ha sido modificado durante la entrada de datos. El comando **Modified**debe utilizarse únicamente en un método de formulario (o una subrutina llamada por un método de formulario).

Atención, este comando sólo devuelve un valor significativo dentro del mismo ciclo de ejecución. Particularmente devuelve **Falso** para todos los eventos de formulario que correspondan al antiguo ciclo de ejecución **_o_During** (On Clicked, On After Keystroke, etc.).

Durante la entrada de datos, un campo se considera modificado si el usuario edita el campo (sin importar si cambia el valor original) y luego pasa a otro campo o hace clic en un control. Note que el hecho de salir de un campo con la ayuda de la tecla Tab no significa que **Modified** devuelva **True**. El campo debe editarse para que **Modified** sea **True**.

Cuando se ejecuta un método, se considera que un campo ha sido modificado si se le asigna un valor (diferente o no).

Nota: Modified siempre devuelve **True** después de la ejecución de los comandos **PUSH RECORD** y **POP RECORD**.

En todos los casos, utilice el comando **Old** apara detectar si el valor del campo en realidad ha sido modificado.

Nota: aunque **Modified** puede aplicarse a todo tipo de campo, si lo utiliza con el comando **Old**, debe tener en cuenta las restricciones que aplican al comando **Old**. Para más detalles, ver la descripción del comando **Old**.

Durante la entrada de datos, generalmente es más fácil realizar operaciones en métodos objeto con ayuda del comando **Evento formulario** que utilizar la función **Modified** en los métodos formulario. Como un método de objeto recibe el evento On Data Change cuando se modifica un campo, el uso de un método de objeto es equivalente a utilizar **Modified** en un método de formulario.

Nota: para funcionar correctamente, el comando **Modified** debe utilizarse únicamente en un método de formulario o en un método llamado por un método de formulario.

Ejemplo 1

El siguiente ejemplo prueba si el campo [Ordenes]Cantidad o el campo [Ordenes]Precio ha sido modificado. Si alguno de los dos ha cambiado, entonces el campo [Ordenes]Total se calcula nuevamente.

```
If ((Modified(</Gen9><span class="rte4d_prm">[Ordenes]Cantidad</span><Gen9>)|(Modified(</Gen9><span class="rte4d_prm">
[Ordenes]Precio</span><Gen9>))
    [Orders]Total :=</Gen9><span class="rte4d_prm">[Ordenes]Cantidad</span><Gen9>*</Gen9><span class="rte4d_prm">
[Ordenes]Precio</span><Gen9>
End if
```

Note que el mismo resultado puede obtenerse utilizando la segunda línea como una subrutina llamada por los métodos de objeto de los campos [Ordenes]Cantidad y [Ordenes]Precio en el evento de formulario **On Data Change**.

Ejemplo 2

Selecciona un registro para la tabla [unaTabla], luego llama múltiples subrutinas que pueden modificar el campo [unaTabla]CampoImportante, pero no guarda el registro. Al final del método principal, puede utilizar el comando **Modified** para determinar si debe guardar el registro:

```
` El registro ha sido seleccionado como registro actual
` Luego realiza acciones utilizando subrutinas
HACER ALGO
HACER ALGO MÁS
NO OLVIDAR HACER ESTO
` ...
` Y luego usted prueba el campo para determinar si el registro debe guardarse
If (Modified([unaTabla]CampoImportante))
   SAVE RECORD([unaTabla])
End if
```

MODIFY RECORD

MODIFY RECORD ({tabla}{;}{*})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a utilizar para entrada de datos o Tabla por defecto, si se omite
*		➡	Ocultar barras de desplazamiento

Descripción

*El comando **MODIFY RECORD** permite al usuario modificar el registro actual de la tabla tabla o de la tabla por defecto si se omite el parámetro tabla. **MODIFY RECORD** carga el registro, si no se ha cargado por el proceso actual y muestra el formulario de entrada actual. Si no hay registro actual, entonces **MODIFY RECORD** no hace nada. **MODIFY RECORD** no afecta la selección actual.*

*El formulario aparece en la ventana del primer plano del proceso. La ventana tiene barras de desplazamiento y una caja de control del tamaño. Si pasa el parámetro opcional * la ventana aparece sin las barras de desplazamiento y sin la caja de control de tamaño.*

*Para utilizar **MODIFY RECORD**, el registro actual debe tener acceso de lectura-escritura y no debe estar bloqueado.*
*Si el formulario contiene botones de navegación entre los registros de la selección, **MODIFY RECORD** le permite al usuario hacer clic en los botones para modificar registros y moverse a otros registros.*

*El registro se guarda (aceptado) si el usuario hace clic en el botón Aceptar o presiona la tecla Intro (teclado numérico) o si se ejecuta el comando **ACCEPT**.*

El registro no se guarda si el usuario hace clic en un botón de tipo CANCELAR o presiona la combinación de teclas (Ctrl-Punto en Windows, Comando-Punto en Macintosh), o si se ejecuta el comando CANCEL.

*Después de llamar **MODIFY RECORD**, la variable sistema OK toma el valor 1 si se acepta el registro y 0 si se cancela.*

***Nota:** el registro permanece en memoria, incluso cuando se cancela, y puede guardarse si se ejecuta **SAVE RECORD** antes de que el puntero del registro actual cambie.*

*Si está utilizando **MODIFY RECORD** y el usuario no realiza ninguna modificación al registro, no se considera que el registro haya sido modificado y aunque acepte el registro no se guardará nuevamente. Las acciones tales como el cambio del valor de variables, la selección de casillas de selección y de botones de radio no clasifican como modificaciones. Únicamente la modificación del valor de un campo, a través de una entrada manual o de un método, hace que el registro se guarde nuevamente.*

Ejemplo

*Ver el ejemplo del comando **ADD RECORD**.*

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si se acepta el registro y 0 si se cancela. La variable OK no toma ningún valor hasta que el registro haya sido validado o anulado.

Old

Old (unCampo) -> Resultado

Parámetro	Tipo		Descripción
unCampo	Campo	➡	Campo del cual devolver el valor anterior
Resultado	Expresión	⮌	Valor original del campo

Descripción

*El comando **Old** devuelve el valor almacenado en campo antes de que se le haya asignado un valor por programación o modificado un durante la entrada de datos.*

Cada vez que cambia el registro actual para una tabla, 4D crea y mantiene en memoria un duplicado de la "imagen" del nuevo registro actual en el momento en que se carga. Cuando usted modifica un registro, trabaja con la imagen actual del registro, no con su duplicado. Esta imagen se borra cuando cambia nuevamente el registro actual.

***Old** devuelve el valor de la imagen duplicada. En otras palabras, para un registro existente, devuelve el valor del campo tal como está guardado en el disco. Si un registro es nuevo, **Old** devuelve el valor vacío por defecto de acuerdo al tipo de campo. Por ejemplo, si campo es un campo Alfa, **Old** devuelve una cadena vacía. Si campo es de tipo numérico, **Old** devuelve cero (0), etc.*

***Old** funciona con campo si el campo ha sido modificado por un método o por el usuario durante la entrada de datos. **Old** puede aplicarse a todos los tipos de campo.*

*Para restaurar el valor original de un campo, asígnele el valor devuelto por **Old**.*

***Nota**: por razones técnicas, **Old** no puede pasarse como parámetro a otros comandos con campos tipo imagen y BLOB. Es necesario pasar el valor por una variable intermedia. Por ejemplo:*

```
  `No escriba (causa error de sintaxis):
$tamano :=BLOB size(Old([LaTabla]ElBlob)) `INCORRECTO
  `Escriba:
$antBLOB:=Old([LaTabla]ElBlob)
$tamano :=BLOB size($antBLOB) `CORRECTO
```

REJECT

REJECT {(unCampo)}

Parámetro	Tipo		Descripción
unCampo	Campo	➡	Campo a rechazar

Descripción

REJECT tiene dos sintaxis. La primera sintaxis no tiene parámetros. En este caso, el comando rechaza la totalidad de la entrada y obliga al usuario a permanecer en el formulario. La segunda sintaxis rechaza solamente el campo y obliga al usuario a permanecer en el campo.

Nota: debe considerar utilizar las herramientas integradas de validación de datos antes de utilizar este comando.

La primera sintaxis de **REJECT** evita que el usuario acepte un registro incompleto. Puede alcanzar el mismo resultado sin utilizar **REJECT**, asocie la tecla Intro con un botón Sin acción y utilice los comandos **ACCEPT** y**CANCEL** para aceptar o cancelar el registro, una vez los campos hayan sido introducidos correctamente. Es recomendable utilizar esta segunda técnica y no la primera sintaxis de **REJECT**.

Si utiliza la primera sintaxis, usted ejecuta **REJECT** para evitar que el usuario acepte un registro, generalmente porque el registro está incompleto o tiene entradas incorrectas. Si el usuario trata de aceptar el registro, la ejecución de **REJECT** evita la aceptación del registro; el registro permanece visualizado en el formulario. El usuario debe continuar con la entrada de datos hasta que el registro sea aceptable o se cancele el registro.

El mejor lugar para el comando **REJECT**, cuando se utiliza esta sintaxis, es el método de objeto de un botón Aceptar asociado a la tecla Intro. De esta forma, la validación ocurre sólo cuando el registro es aceptado y el usuario no puede forzar la validación presionando la tecla Enter.

La segunda sintaxis de **REJECT** se ejecuta con el parámetro campo. En este caso, el cursor permanece en el área de entrada del campo, que obliga al usuario a introducir un valor correcto.
Con esta sintaxis, el comando **REJECT** debe obligatoriamente ser llamado en el evento formulario On Data Change. Debe colocar esta sintaxis del comando **REJECT** en el método de formulario o en el método de objeto del área de entrada. Si está utilizando **REJECT** para el formulario detallado de un subformulario para una tabla, colóquelo en el método de formulario o método de objeto para el formulario detallado. Este comando no tiene efecto en campos de subformularios.

Puede utilizar **HIGHLIGHT TEXT** para seleccionar los datos en el campo que están siendo rechazados.

Ejemplo 1

El siguiente ejemplo es sobre un registro de transacción bancaria que ilustra la primera sintaxis de **REJECT** utilizada en el método de objeto de un botón Aceptar. La tecla Enter está definida como un equivalente del botón. Esto significa que incluso si el usuario presiona la tecla Enter para aceptar el registro, el método de objeto del botón se ejecutará. Si la transacción es un cheque, entonces debe haber un número de cheque. Si no hay un número de cheque, se rechaza la validación:

```
Case of
   :(([Operacion]Transaccion="Cheque") & ([Operacion]Número Cheque=""))  ` Si es un cheque sin número.
      ALERT("Por favor introduzca el número del cheque.")  ` Alerta del usuario
      REJECT  ` Rechazar la entrada
      GOTO OBJECT([Operacion]Número Cheque)  ` Ir al campo Número Cheque
End case
```

Ejemplo 2

El siguiente ejemplo es parte de un método de objeto para un campo [Employees]Salary. El método de objeto prueba el campo Employees]Salary y rechaza el campo si el valor es menor que 10 000 EUR. Puede efectuar la misma operación especificando un valor mínimo para el campo en el editor de formularios:

```
Case of
   :(Form event=On Data Change)
      If([Employees]Salary<10000)
         ALERT("Salary must be greater than $10,000")
         REJECT([Employees]Salary)
      End if
End case
```

_o_ADD SUBRECORD

_o_ADD SUBRECORD (subtabla ; form {; *})

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla a utilizar para la entrada de datos
form	Cadena	➡	Formulario a utilizar para la entrada de datos
*		➡	Ocultar barras de desplazamiento

Nota de compatibilidad

A partir de la versión 11 de 4D no se soportan las subtablas. Un mecanismo de seguridad asegura el funcionamiento de este comando en bases convertidas; sin embargo, se recomienda reemplazar las subtablas con tablas relacionadas estándar.

_o_MODIFY SUBRECORD

_o_MODIFY SUBRECORD (subtabla ; formulario {; *})

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla a utilizar para la entrada de datos
formulario	Cadena	➡	Formulario a utilizar para la entrada de datos
*		➡	Ocultar barras de desplazamiento

Nota de compatibilidad

A partir de la versión 11 de 4D, las subtablas no son soportadas. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en las bases de datos convertidas, sin embargo es recomendable reemplazar las subtablas por las tablas relacionadas estándar.

Estructura

- *Comandos de acceso a estructura*
- *CREATE INDEX*
- *DELETE INDEX*
- *EXPORT STRUCTURE*
- *Field*
- *Field name*
- *Get external data path*
- *GET FIELD ENTRY PROPERTIES*
- *GET FIELD PROPERTIES*
- *Get last field number*
- *Get last table number*
- *GET MISSING TABLE NAMES*
- *GET RELATION PROPERTIES*
- *GET TABLE PROPERTIES*
- *IMPORT STRUCTURE*
- *Is field number valid*
- *Is table number valid*
- *PAUSE INDEXES*
- *REGENERATE MISSING TABLE*
- *RELOAD EXTERNAL DATA*
- *RESUME INDEXES*
- *SET EXTERNAL DATA PATH*
- *SET INDEX*
- *Table*
- *Table name*

Comandos de acceso a estructura

Los comandos de este tema devuelven la descripción de la estructura de la base. Permiten conocer el número de tablas, el número de campos en cada tabla, los nombres de las tablas y campos, así como el tipo y propiedades de cada campo. Los comandos de utilidades pueden utilizarse para detectar y regenerar tablas perdidas para recuperar datos "fantasmas".

Determinar la estructura precisa de la base es muy útil cuando desarrolla y utiliza grupos de métodos de proyecto y formularios que pueden copiarse en diferentes bases.

La posibilidad de leer la estructura de la base permite desarrollar y utilizar código portable.

***Nota:** puede crear y modificar los campos y tablas 4D por programación utilizando los comandos del motor SQL integrado de 4D, como **CREATE TABLE** o **ALTER TABLE**. Para más información, consulte el manual "**Manual de SQL**".*

Contar las tablas y campos

*Es posible borrar las tablas y los campos 4D. Esta posibilidad se debe tener en cuenta en los algoritmos utilizados para contar tablas y campos. Es necesario utilizar algoritmos combinando los comandos **Get last table number** y **Get last field number**, así como también **Is table number valid** y **Is field number valid**. El siguiente es un ejemplo de este tipo de algoritmo:*

```
For($thetable;1;Get last table number)
   If(Is table number valid($thetable))
      For($thefield;1;Get last field number($thetable))
         If(Is field number valid($thetable;$thefield))
            ... `El campo existe y es válido
         End if
      End for
   End if
End for
```

CREATE INDEX

CREATE INDEX (tabla ; arrayCampos ; tipoIndice ; nombreIndice {; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual crear un índice
arrayCampos	Array puntero	➡	Puntero(s) a el/los campo(s) a indexar
tipoIndice	Entero largo	➡	Tipo de índice a crear: -1 = Palabras claves, 0 = por defecto, 1 = B-Tree estándar, 3 = B-Tree cluster
nombreIndice	Texto	➡	Nombre del índice a crear
*	Operador	➡	Si pasa = indexación asincrónica

Descripción

*El comando **CREATE INDEX** permite crear:*

- *Un índice estándar en uno o más campos (índice compuesto) o*
- *Un índice de palabras claves en un campo.*

El índice se crea para la tabla laTabla utilizando uno o más campos designados por el array de punteros arrayCampos. Este array contiene una sola línea si quiere crear un índice simple y dos o más cuando quiere crear un índice compuesto (excepto en el caso de un índice de palabras claves). En el caso de los índices compuestos, el orden de los campos en el array es importante durante la construcción del índice.

*El parámetro tipoIndice permite definir el tipo de índice a crear. Puede pasar una de las siguientes constantes, que se encuentran en el tema "**Tipo de índice**":*

Constante	Tipo	Valor	Comentario
Cluster BTree index	Entero largo	3	Índice de tipo B-Tree utilizando clusters. Este tipo de índice se optimiza cuando el índice contiene pocas palabras claves, es decir cuando los mismos valores se presentan con frecuencia en los datos.
Default index type	Entero largo	0	4D define el tipo de índice (excepto los índices de palabras claves) que es el más optimizado en función del contenido del campo.
Keywords index	Entero largo	-1	Permite la indexación palabra por palabra del contenido del campo. Este tipo de índice sólo puede utilizarse con campos de tipo Texto, Alfa e Imagen. Atención los índices de palabras claves no pueden ser compuestos
Standard BTree index	Entero largo	1	Índice de tipo B-Tree clásico. Este tipo de índice multi propósito se utiliza en las versiones anteriores de 4D

***Nota:** un índice B-Tree asociado a un campo de tipo texto almacena como máximo los primeros 1024 caracteres del campo. Por lo tanto en este contexto, las búsquedas en las cadenas que contienen más de 1024 caracteres fallarán.*

*Pase en nomIndice el nombre del índice a crear. Es necesario dar nombres a los índices si varios índices de diferentes tipos pueden asociarse a un mismo campo y si prefiere poder borrarlos individualmente con la ayuda del comando **DELETE INDEX**. Si el índice nomIndice ya existe, el comando no hace nada.*

*El parámetro opcional *, cuando se pasa, permite efectuar la indexación en modo asincrónico. En este modo, el método original continua su ejecución después de la llamada del comando, sin importar si la indexación ha terminado o no.*
*Si el comando **CREATE INDEX** encuentra registros bloqueados, estos no se indexarán y el comando esperará a que se desbloqueen.*

Si ocurre un problema durante la ejecución del comando (campo no indexable, intento de creación de un índice de palabras claves de más de un campo, etc.), se genera un error. Este error puede interceptarse utilizando un método de gestión de errores.

Ejemplo 1

Creación de dos índices estándar en los campos "Apellido" y "Teléfono" de la tabla [Clientes]:

```
ARRAY POINTER(arrayPtrCampo;1)
fieldPtrArr{1}:=->[Clientes]Apellido
CREATE INDEX([Clientes];arrayPtrCampo;Standard BTree Index;"CustLNameIdx")
fieldPtrArr{1}:=->[Clientes]Telefono
CREATE INDEX([Clientes];arrayPtrCampo;Standard BTree Index;"CustTelIdx")
```

Ejemplo 2

Creación de un índice de palabras claves en el campo "Observaciones" de la tabla [Clientes]:

```
ARRAY POINTER(arrayPtrCampo;1)
fieldPtrArr{1}:=->[Clientes]Observaciones
CREATE INDEX([Clientes];arrayPtrCampo;Keywords Index;"CustObsIdx")
```

Ejemplo 3

Creación de un índice compuesto en los campos "Ciudad" y "CodigoPostal" de la tabla [Clientes]:

```
ARRAY POINTER(arrayPtrCampo;2)
fieldPtrArr{1}:=->[Clientes]Ciudad
fieldPtrArr{2}:=->[Clientes]CodigoPostal
CREATE INDEX([Clientes];arrayPtrCampo;Standard BTree Index;"CityZip")
```

DELETE INDEX

DELETE INDEX (Ptrcamp | nomIndex {; *})

Parámetro	Tipo		Descripción
Ptrcamp | nomIndex	Puntero, Cadena	➡	Puntero al campo del cual borrar los índices o Nombre del índice a borrar
*	Operador	➡	Si se pasa = operación asincrónica

Descripción

El comando **DELETE INDEX** se utiliza para borrar uno o más índices existentes en la base.
Puede pasar en parámetro un puntero o un campo o el nombre de un índice en el parámetro:

- Si pasa un puntero a un campo (ptrcampo), todos los índices asociados al campo serán borrados. Puede tratarse de índices de palabras claves o de índices estándar. Sin embargo, si el campo está incluido en uno o más índices compuestos, no se borrarán (debe pasar un nombre de índice).
- Si pasa el nombre de un índice (nomIndex), sólo se borrará el índice designado. Puede tratarse de índices de palabras claves, índices estándar o índices compuestos.

El parámetro opcional *, cuando se pasa, permite efectuar la desindexación en modo asincrónico. En este modo, el método de origen continúa su ejecución después de la llamada al comando, sin importar si la eliminación del índice a terminado o no.
Si no existe un índice correspondiente a **Ptrfcamp** o nomIndex, el comando no hace nada.

Ejemplo

Este ejemplo ilustra las dos sintaxis del comando:

```
`Eliminación de todos los índices relacionados con el campo Apellido
DELETE INDEX(->[Clientes]Apellido)
`Eliminación del índice llamado "CPCiudad"
DELETE INDEX("CPCiudad")
```

EXPORT STRUCTURE

EXPORT STRUCTURE (estructuraXML)

Parámetro	Tipo		Descripción
estructuraXML	Variable texto	⇐	Exportación de la definición XML de la estructura de la base 4D

Descripción

*El comando **EXPORT STRUCTURE** exporta, en estructuraXML, la definición de la estructura de la base 4D actual al formato XML. Este comando utiliza los mismos mecanismos que el comando de menú **Exportar > Definición de estructura al archivo XML...** que se encuentra en la interfax del modo Diseño de 4D (ver **Exportar e importar las definiciones de estructura**).*

En estructuraXML, pase la variable texto destinada a almacenar la definición de la estructura. La definición exportada incluye las tablas, los campos, los índices y las relaciones, así como también sus atributos y todas las características necesarias para obtener una descripción completa de la estructura. Los elementos invisibles se exportan con el atributo correspondiente. Sin embargo, los elementos eliminados no se exportan.

*La "gramática" interna de las definiciones de estructura 4D se documenta por medio de los archivos DTD, también utilizados para la validación de los archivos XML. Los archivos DTD utilizados por 4D se agrupan en la carpeta **DTD**, que se encuentra junto a la aplicación 4D. Los archivos **base_core.dtd** y **common.dtd** se utilizan para definiciones de la estructura. Para obtener más información, puede consultar estos archivos junto con los comentarios que contienen.*

Ejemplo

Usted desea exportar la estructura de la base a un documento de texto:

```
C_TEXT($vTStruc)
EXPORT STRUCTURE($vTStruc)
TEXT TO DOCUMENT("myStructure.xml";$vTStruc)
```

Field

Field (numTabla ; numCamp) -> Resultado

Parámetro	Tipo		Descripción
numTabla	Entero largo	→	Número de tabla
numCamp	Entero largo	→	Número de campo
Resultado	Puntero	↩	Puntero de campo

Field (ptrCamp) -> numCampo

Parámetro	Tipo		Descripción
ptrCamp	Puntero	→	Puntero del campo
numCampo	Entero largo	↩	Número de campo

Descripción

*El comando **Field** tiene dos sintaxis:*

- *Si pasa un número de tabla en numTabla y un número de campo en numCampo, **Field** devuelve un puntero al campo.*
- *Si pasa un puntero a un campo en ptrCamp, **Field** devuelve el número del campo.*

Ejemplo 1

El siguiente ejemplo asigna la variable campPtr a un puntero al segundo campo en la tercera tabla:

```
CampPtr:=Field(3;2)
```

Ejemplo 2

*Si pasa campPtr (un puntero al segundo campo de una tabla) a **Field** devuelve el valor 2. La siguiente línea asigna el valor 2 a campNum:*

```
campNum:=Field(campPtr)
```

Ejemplo 3

En el siguiente ejemplo, la variable campNum es igual al número del campo de [Tabla3]Campo2:

```
campNum:=Field(->[Tabla3]Campo2)
```

Field name

Field name (campPtr | tablaNum {; numCamp}) -> Resultado

Parámetro	Tipo		Descripción
campPtr | tablaNum	Puntero, Entero largo	➡	Puntero a un campo o número de tabla
numCamp	Entero largo	➡	Número de campo si se pasa un número de tabla como primer parámetro
Resultado	Cadena	➲	Nombre del campo

Descripción

*El comando **Field name** devuelve el nombre del campo cuyo puntero se pasa en campPtr o cuyos números de tabla y de campos se pasan en tablaNum y campNum.*

Ejemplo 1

Este ejemplo asigna el segundo elemento del array campArray{1} al nombre del segundo campo en la primera tabla. campArray es un array de dos dimensiones:

```
campArray{1}{2}:=Field name(1;2)
```

Ejemplo 2

Este ejemplo asigna al segundo elemento del array campArray{1} el nombre del campo [MiTabla]MiCampo. campArray es un array de dos dimensiones:

```
campArray{1}{2}:=Field name(->[MiTabla]MiCampo)
```

Ejemplo 3

Este ejemplo muestra una alerta. Este método pasa un puntero a un campo:

```
ALERT("El número del campo "+Field name($1)+" de la tabla "
+Table name(Table($1))+" debe ser de más de cinco caracteres.")
```

Get external data path

Get external data path (elCampo) -> resultado

Parámetro	Tipo		Descripción
elCampo	Texto, BLOB, Imagen	➡	Campo del cual obtener el lugar de almacenamiento
resultado	Texto	⮌	Ruta de acceso completa del archivo de almacenamiento externo

Descripción

*El comando **Get external data path** devuelve la ruta de acceso completa del archivo de almacenamiento externo de datos del campo pasado en el parámetro elCampo, para el registro actual. Debe pasar en el parámetro elCampo campos de tipo Texto, BLOB o Imagen. El comando devuelve la ruta de acceso del archivo de almacenamiento si el archivo no existe más o no es accesible.*

Más específicamente, este comando le permite copiar nuevamente el archivo externo.

***Nota**: para obtener más información sobre el almacenamiento externo de datos, consulte el manual de Diseño.*

Este comando devuelve una cadena vacía en los siguiente casos:

- *El campo no se guarda fuera del archivo de datos.*
- *El campo tiene un valor Null (y no contiene ruta de acceso),*
- *El comando se ejecuta desde un 4D remoto.*

GET FIELD ENTRY PROPERTIES

GET FIELD ENTRY PROPERTIES (ptrCamp|numTabla {; numCamp}; lista ; obligatorio ; noEditable ; noModificable)

Parámetro	Tipo		Descripción
ptrCamp|numTabla	Puntero, Entero largo	➡	Puntero del campo o número de tabla
numCamp	Entero largo	➡	Número de campo si el número de tabla se pasa como primer parámetro
lista	Cadena	⬅	Nombre de la lista asociada o cadena vacía
obligatorio	Booleano	⬅	True = Obligatorio, False = Opcional
noEditable	Booleano	⬅	True = No editable, False = Editable
noModificable	Booleano	⬅	True = No modificable, False = Modificable

Descripción

El comando ***GET FIELD ENTRY PROPERTIES*** *devuelve las propiedades de entrada de datos para el campo especificado por numTabla y numCamp o por ptrCamp.*

Puede pasar:

- *números de tabla y de campo en numTabla y numCamp, o*
- *un puntero al campo en ptrCamp.*

Nota: *este comando devuelve las propiedades definidas a nivel de la ventana de estructura de la base. Propiedades similares pueden definirse a nivel de los formularios.*

Una vez ejecutado el comando:

- *El parámetro list devuelve el nombre de la lista asociada al campo (si la hay). Es posible asociar una lista a los siguientes tipos de campos: Alfa, Texto, Numérico, Entero, Entero largo, Fecha, Hora y Booleano.*

Si ninguna lista está asociada al campo o si el tipo del campo no permite la asociación de listas, se devuelve una cadena vacía ("").

- *El parámetro obligatorio devuelve True si el campo es "Obligatorio"; de lo contrario False. El atributo "obligatorio" puede asociarse a todo tipo de campos, excepto BLOB.*
- *El parámetro noEditable devuelve True si el campo dispone del atributo "No editable", de lo contrario False. Un campo no editable únicamente puede leerse, no acepta entrada de datos. El atributo "No modificable" puede asociarse a campos de todos los tipos, excepto BLOB.*
- *El parámetro noModificable devuelve True si el campo es "No modificable", de lo contrario False. Un campo no modificable acepta sólo una entrada y no puede ser modificado. El atributo "No modificable" puede ser definido para todos los tipos de campos, excepto BLOB.*

GET FIELD PROPERTIES

GET FIELD PROPERTIES (campPtr | tablaNum {; numCamp}; campTipo {; campLong {; indexado {; unico {; invisible}}}})

Parámetro	Tipo		Descripción
campPtr | tablaNum	Puntero, Entero largo	➡	Puntero de campo o Número de tabla
numCamp	Entero largo	➡	Número de campo si se pasa un número de tabla
campTipo	Entero largo	⬅	Tipo de campo
campLong	Entero largo	⬅	Longitud del campo, si es alfanumérico
indexado	Booleano	⬅	True = Indexado, False = No indexado
unico	Booleano	⬅	True = único, False = No único
invisible	Booleano	⬅	True = Invisible, False = Visible

Descripción

El comando **GET FIELD PROPERTIES** devuelve información sobre el campo designado por campPtr o por tablaNum y campNum.

Puede pasar:

- los números de tabla y de campo en tablaNum y campNum, o
- un puntero al campo en campPtr.

Después de la llamada:

- campTipo devuelve el tipo del campo. El parámetro variable campTipo recibe uno de los valores predefinidos por las constantes de 4D (tema **Tipos de campos y variables**):

Constante	Tipo	Valor
Is alpha field	Entero largo	0
Is BLOB	Entero largo	30
Is Boolean	Entero largo	6
Is date	Entero largo	4
Is float	Entero largo	35
Is integer	Entero largo	8
Is integer 64 bits	Entero largo	25
Is longint	Entero largo	9
Is object	Entero largo	38
Is picture	Entero largo	3
Is real	Entero largo	1
Is subtable	Entero largo	7
Is text	Entero largo	2
Is time	Entero largo	11

- El parámetro campLong devuelve la longitud del campo, si el campo es alfanumérico (es decir, campTipo=Is Alpha Field). El valor de campField no es significativo para los otros tipos de campo.
- El parámetro indexado devuelve True si el campo está indexado, de lo contrario False. El valor de indexado es significativo únicamente para campos de tipo Alfanumérico, Entero, Entero largo, Real, Fecha, Hora y Booleano.
- El parámetro unico devuelve True si el campo está definido como "único", de lo contrario False.
- El parámetro invisible devuelve True si el campo está definido como "Invisible", de lo contrario False. El atributo invisible puede ser utilizado para ocultar un campo en el editor estándar de 4D (etiquetas, gráficos...).

Ejemplo 1

En este ejemplo, las variables vTipo, vLong, vIndex, vUnico y vInvisible toman por valor las propiedades del tercer campo de la primera tabla:

```
GET FIELD PROPERTIES(1;3;vTipo;vLong;vIndex;vUnico;vInvisible)
```

Ejemplo 2

Este ejemplo recupera en las variables vTipo, vLong, vIndex, vUnico y vInvisible las propiedades del campo [Tabla3]Campo2:

```
GET FIELD PROPERTIES(->[Tabla3]Campo2;vTipo;vLong;vIndex;vUnico;vInvisible)
```

Get last field number

Get last field number (numTabla | ptrTabla) -> Resultado

Parámetro	Tipo		Descripción
numTabla | ptrTabla	Entero largo, Puntero	➡	Número de tabla o puntero a una tabla
Resultado	Entero largo	⮌	Número de campo más alto en la tabla

Descripción

*El comando **Get last field number** devuelve el número de campo más alto de los campos en la tabla cuyo número o puntero se pasa en numTabla o ptrTabla.*

*Los campos están numerados en el orden en el cual fueron creados. Si ningún campo ha sido borrado de la tabla, este comando devuelve el número de campos que contiene la tabla. En el caso de bucles interactivos sobre los números de campo de la tabla, debe utilizar el comando **Is field number valid** con el fin de verificar que el campo no ha sido eliminado.*

Ejemplo

El siguiente método de proyecto crea el array asCampos, con los nombres de los campos de la tabla cuyo puntero se recibe como primer parámetro:

```
$vlTabla:=Table($1)
ARRAY STRING(31;asCampos;Get last field number($vlTabla))
For($vlCampo;Size of array(asCampos);1;-1)
   If(Is field number valid($vlTabla;$vlCampo)
      asCampos{$vlCampo}:=Field name($vlTabla;$vlCampo)
   Else
      DELETE FROM ARRAY(asCampos;$vlCampo)
   End if
End for
```

Get last table number

Get last table number -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Número de tabla más alto en la base

Descripción

Get last table number *devuelve el número de tablas en la base. Las tablas están numeradas en el orden en el cual fueron creadas. Si ninguna tabla ha sido borrada de la base, el comando devuelve el número de tablas presentes en la base. En el caso de bucles iterativos sobre los números de tablas de la base, debe utilizar el comando* ***Is table number valid*** *para verificar que la tabla no haya sido borrada.*

Ejemplo

El siguiente ejemplo inicializa los elementos del array asTablas, con los nombres de las tablas definidos en la base. Este array puede utilizarse como lista desplegable (o pestañas, área de desplazamiento, etc.), para mostrar en un formulario la lista de tablas de la base:

```
ARRAY STRING(31;asTablas;Get last table number)
If(Get last table number>0)  `Si la Base de datos no tiene tablas
   For($vlTablas;Size of array(asTablas);1;-1)
      If(Is table number valid($vlTablas))
         asTablas{$vlTablas}:=Table name($vlTablas)
      Else
         DELETE FROM ARRAY(asTablas;$vlTablas)
      End if
   End for
End if
```

GET MISSING TABLE NAMES

GET MISSING TABLE NAMES (tabfalt)

Parámetro	Tipo		Descripción
tabfalt	Array texto	⇐	Nombres de las tablas faltantes en la base

Descripción

*El comando **GET MISSING TABLE NAMES** devuelve los nombres de todas las tablas faltantes de la base en el array tabFalt.*
Las tablas faltantes son tablas cuyos datos están presentes en el archivo de datos pero que no existen a nivel de la estructura actual. Esto puede suceder cuando un archivo de datos se abre con versiones diferentes de la estructura.
Por lo general, el escenario es el siguiente:

- *el desarrollador ofrece una estructura con las tablas A, B y C,*
- *el usuario añade tablas personalizadas D y E, utilizando, por ejemplo, los comandos integrados de 4D, y guarda los datos en estas tablas,*
- *El desarrollador ofrece una nueva versión de la estructura, que no contiene las tablas D y E.*
  *En este caso, la versión usuario de la base aún contiene los datos de las tablas D y E, pero no son accesibles. El comando **GET MISSING TABLE NAMES** devolverá los nombres "D" y "E".*

*Una vez haya identificado las tablas faltantes de la base, puede reactivarlas vía el comando **REGENERATE MISSING TABLE**.*
***Nota:** los datos de las tablas faltantes se borran cuando se compacta el archivo de datos (si las tablas no han sido regeneradas).*

GET RELATION PROPERTIES

GET RELATION PROPERTIES (ptrCamp|numTabla {; numCamp}; tablaUno ; campUno {; discriminante {; autoUno {; autoMuchos}}})

Parámetro	Tipo		Descripción
ptrCamp|numTabla	Puntero, Entero largo	➡	Puntero de campo o número de tabla
numCamp	Entero largo	➡	Número de campo si se pasa un número de tabla como primer parámetro
tablaUno	Entero largo	⬅	Número de la tabla Uno ó 0 si no se define ninguna relación desde el campo
campUno	Entero largo	⬅	Número de campo Uno ó 0 si no se define ninguna relación desde el campo
discriminante	Entero largo	⬅	Número de campo discriminante o 0 si ningún campo discriminante
autoUno	Booleano	⬅	True = Relación uno automática, False = Relación uno manual
autoMuchos	Booleano	⬅	True = Relación unos a muchos automática, False = Relación unos a muchos manual

Descripción

*El comando **GET RELATION PROPERTIES** devuelve las propiedades de la relación (si la hay) que comienza del campo fuente definido por numTabla y numCamp o por ptrCamp.*

Puede pasar:

- *Números de tabla y de campo en numTabla y numCamp,*
- *O un puntero al campo en ptrCamp.*

Una se haya ejecutado el comando:

- *Los parámetros tablaUno y campoUno contienen respectivamente el número de la tabla y del campo hacia los cuales apunta la relación (del campo fuente). Si ninguna relación comienza en el campo, este parámetro devuelve 0.*
- *El parámetro discriminante contiene el número del campo discriminante (de la tabla objetivo) definido dentro de esta relación. Si no se ha definido un campo discriminante en esta relación, o si ninguna relación parte del campo fuente, este parámetro devuelve 0.*
- *Los parámetro autoUno y autoMuchos devuelven **True** si, respectivamente, las opciones "Relación uno a muchos automática" y "Relación muchos a uno automática" se han seleccionado para esta relación; de lo contrario, devuelven **False**.*

***Nota:** los parámetros autoUno y autoMuchos también devolverán **True** si ninguna relación parte del campo fuente (en este caso devuelven valores no significativos.). El valor de los parámetros tablaUno y campUno permiten asegurarse de que una relación existe.*

GET TABLE PROPERTIES

GET TABLE PROPERTIES (PtrTabla|numTabla ; invisible {; trigGuardarNuevo {; trigGuardaReg {; trigBorrarReg {; trigCargReg}}}})

Parámetro	Tipo		Descripción
PtrTabla|numTabla	Puntero, Entero largo	➡	Puntero de tabla o número de tabla
invisible	Booleano	⬅	True = Invisible, False = Visible
trigGuardarNuevo	Booleano	⬅	True = Trigger "On saving new record" activado; de lo contrario, False
trigGuardaReg	Booleano	⬅	True = Trigger "On saving an existing record" activado; de lo contrario, False
trigBorrarReg	Booleano	⬅	True = Trigger "On deleting a record" activado; de lo contrario, False
trigCargReg	Booleano	⬅	*** No usado (obsoleto) ***

Descripción

*El comando **GET TABLE PROPERTIES** devuelve las propiedades de la tabla pasada por ptrTabla o numTabla. Puede pasar en el primer parámetro el número de tabla o puntero de la tabla.*

Una vez ejecutado el comando:

- *El parámetro invisible devuelve True si el atributo "Invisible" ha sido definido para la tabla, de lo contrario False. El atributo invisible permite ocultar la tabla en los editores estándar de 4D (etiquetas, gráficos...).*
- *El parámetro trigGuardarNuevo devuelve True si el trigger "Al guardar un registro nuevo" se ha activado para la tabla, de lo contrario False.*
- *El parámetro trigGuardaReg devuelve True si el trigger "Al guardar un registro existente" se ha activado para la tabla, de lo contrario False.*
- *El parámetro trigBorrarReg devuelve True si el trigger "Al borrar un registro" se ha activado para esta tabla, de lo contrario False.*

IMPORT STRUCTURE

IMPORT STRUCTURE (estructuraXML)

Parámetro	Tipo		Descripción
estructuraXML	Texto	➡	Definición XML de la estructura de la base 4D

Descripción

*El comando **IMPORT STRUCTURE** importa, en la base actual, la definición XML de la estructura de la base 4D pasada en el parámetro estructuraXML.*

El parámetro estructuraXML debe contener una definición válida de estructura 4D en formato XML. Hay varias maneras de obtener una definición de estructura válida:

- *Ejecutar el comando **EXPORT STRUCTURE**,*
- *Seleccionar el comando de menú **Exportar > Definición de estructura al archivo XML...** disponible en la interfaz del modo Diseño de 4D (ver **Exportar e importar las definiciones de estructura**),*
- *Crear o modificar un archivo XML personalizado basado en los DTD públicos que se encuentran en la carpeta "DTD" de la aplicación 4D.*

La definición de estructura importada se agrega a la estructura que ya está abierta y se muestra en el editor de estructura estándar de 4D entre las tablas existentes (si las hubiera). Si una tabla importada tiene el mismo nombre que una local, se genera un error y la operación de importación se cancela.

Puede crear una nueva base mediante la importación de una definición de estructura en una base vacía.

Se genera un error si la estructura está en modo compilado y/o de sólo lectura.

Este comando no se puede llamar desde una aplicación 4D que funciona en modo remoto.

Ejemplo

Usted desea importar una definición de estructura guardada en la base actual:

```
$struc:=Document to text("c:\\4DStructures\\Employee.xml")
IMPORT STRUCTURE($struc)
```

Is field number valid

Is field number valid (numTabla | ptrTabla ; numCamp) -> Resultado

Parámetro	*Tipo*		*Descripción*
numTabla | *ptrTabla*	*Entero largo, Puntero*	➡	*Número de tabla o Puntero a una tabla*
numCamp	*Entero largo*	➡	*Número de campo*
Resultado	*Booleano*	⮌	*True = el campo existe en la tabla False = el campo no existe en la tabla*

Descripción

*El comando **Is field number valid** devuelve True si el campo cuyo número se pasa en el parámetro numCamp existe en la tabla cuyo número o puntero se pasa en el parámetro numTabla o ptrTabla. Si el campo no existe, el comando devuelve False. Recuerde que el comando devuelve False si la tabla que contiene el campo se encuentra en la Papelera del Explorador.*

Este comando permite detectar las eventuales eliminaciones de campos, que crean rupturas en la secuencia de números de los campos.

Is table number valid

Is table number valid (numTabla) -> Resultado

Parámetro	*Tipo*		*Descripción*
numTabla	*Entero largo*	⇒	*Número de tabla*
Resultado	*Booleano*	⇐	*True = la tabla existe en la base, False = la tabla no existe en la base*

Descripción

*El comando **Is table number valid** devuelve True si la tabla cuyo número se pasa en el parámetro numTabla existe en la base, de lo contrario False. Recuerde que el comando devuelve False si la tabla está en la Papelera del Explorador.*

Este comando permite detectar las eventuales eliminaciones de tablas, que crean rupturas en la secuencia de números de las tablas.

PAUSE INDEXES

PAUSE INDEXES (laTabla)

Parámetro	Tipo		Descripción
laTabla	Tabla	➡	Tabla para la cual detener los índices

Descripción

*El comando **PAUSE INDEXES** desactiva temporalmente todos los índices de laTabla, excepto el índice de la llave primaria .*

*Los índices no se eliminan físicamente de los datos (archivo .4DIndx) o de la estructura de la base (_USER_INDEXES, ver **Tablas sistema**), pero son inválidos y por lo tanto no se actualizan. Cuando los índices están desactivados, todas las operaciones realizadas en laTabla (búsquedas, ordenaciones, adiciones, modificaciones y eliminaciones de registros) ya no utilizan los índices.*

Este comando es especialmente útil cuando se va a importar o modificar grandes cantidades de datos en tablas que tienen varios índices. Como 4D debe actualizar los índices cada vez que un registro se valida, la operación podría tomar una cantidad considerable de tiempo. Desactivar los índices de antemano permite acelerar significativamente la operación.

*Para reactivar los índices después de que termina la operación, puede llamar al comando **RESUME INDEXES** para laTabla.*

***Nota:** puede obtener un resultado similar utilizando los comandos **CREATE INDEX** y **DELETE INDEX**, pero con diferencias notables:*

- *es necesario llamar a **DELETE INDEX** / **CREATE INDEX** para cada índice en laTabla.*
- *llamar a los comandos **DELETE INDEX** / **CREATE INDEX** cambia el número interno del índice, lo que no ocurre con **PAUSE INDEXES** / **RESUME INDEXES**. Cambiar el número de índice generará una reindexación automática de los datos si el conjunto de datos cambia.*

*Si llama al comando **PAUSE INDEXES** para una tabla y luego sale de la base sin haber llamado al comando **RESUME INDEXES** para esta tabla, todos los índices de la tabla se reconstruyen automáticamente cuando se reinicie la base.*

***Nota**: este comando no se puede ejecutar desde un 4D remoto.*

Ejemplo

Ejemplo de método de importación masivo de datos:

```
PAUSE INDEXES([Articles])
IMPORT DATA("HugeImport.txt") //Importando
RESUME INDEXES([Articles])
```

REGENERATE MISSING TABLE

REGENERATE MISSING TABLE (nomTabla)

Parámetro	Tipo		Descripción
nomTabla	Texto	➡	Nombre de tabla faltante a regenerar

Descripción

El comando **REGENERATE MISSING TABLE** reconstruye la tabla faltante cuyo nombre se pasa en el parámetro nomTabla. Cuando se reconstruye una tabla faltante, se vuelve visible en el editor de estructura y sus datos son accesibles nuevamente.

Las tablas faltantes son tablas cuyos datos están presentes en el archivo de datos pero que no existen a nivel de la estructura. Puede identificar las tablas faltantes que puedan estar presentes utilizando el comando **GET MISSING TABLE NAMES**.

Si la tabla designada por el parámetro nomTabla no es una tabla faltante de la base, el comando no hace nada.

Ejemplo

Este método regenera todas las tablas faltantes eventualmente presentes en la base:

```
ARRAY TEXT($arrMissingTables;0)
GET MISSING TABLE NAMES($arrMissingTables)
$SizeArray:=Size of array($arrMissingTables)
If($SizeArray#0)
 // Llena el array con los nombres de todas las tablas en la base
   ARRAY TEXT(arrTables;Get last table number)
   If(Get last table number>0) //Si hay tablas
      For($vlTables;Size of array(arrTables);1;-1)
         If(Is table number valid($vlTables))
            arrTables{$vlTables}:=Table name($vlTables)
         Else
            DELETE FROM ARRAY(arrTables;$vlTables)
         End if
      End for
   End if
   For($i;1;$SizeArray)
      If(Find in array(arrTables;$arrMissingTables{$i})=-1)
         CONFIRM("Regenerar la tabla"+$arrMissingTables{$i}+"?")
         If(OK=1)
            REGENERATE MISSING TABLE($arrMissingTables{$i})
         End if
      Else
         ALERT("Imposible regenerar la tabla "+$arrMissingTables{$i}+" porque ya hay una tabla con este nombre en la base.")
      End if
   End for
Else
   ALERT("No ha tablas a regenerar.")
End if
```

RELOAD EXTERNAL DATA

RELOAD EXTERNAL DATA (elCampo)

Parámetro	***Tipo***		***Descripción***
elCampo	*Texto, BLOB, Imagen, Objeto*	➡	*Campo para el cual recargar los datos*

Descripción

*El comando **RELOAD EXTERNAL DATA** permite recargar en memoria el contenido de un archivo de almacenamiento externo asociado a un campo de tipo BLOB, Imagen, Texto u Objeto.*

Este comando es útil cuando el campo de un registro ya cargado en memoria es modificado en el disco por otra aplicación (los archivos de almacenamiento externo de los campos siempre son accesibles en escritura). Por ejemplo, una imagen utilizada en un campo Imagen puede ser modificada por un editor gráfico y luego de guardarse en el disco.

*A continuación debe recargar los datos utilizando el comando **RELOAD EXTERNAL DATA** para actualizar los contenidos del campo si se muestra en un formulario.*

***Nota**: el comando **RELOAD EXTERNAL DATA** sólo funciona 4D local o 4D Server. No es posible recargar individualmente un campo con 4D en modo remoto. En este contexto, es necesario recargar todos los registros (utilizando el comando **LOAD RECORD** por ejemplo).*

RESUME INDEXES

RESUME INDEXES (tabla {; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual reactivar los índices
*	Operador	➡	Si se pasa = indexación asíncrona

Descripción

*El comando **RESUME INDEXES** reactiva todos los índices de la tabla cuando se han detenido anteriormente utilizando el comando **PAUSE INDEXES**. Si los índices de tabla no se han detenido, el comando no hace nada.*

En la mayoría de los casos, la ejecución de este comando activa la reconstrucción de los índices de tabla.
*Si pasa el parámetro opcional *, la reconstrucción de los índices se realiza en modo asíncrono. Esto significa que el método de llamada al comando continúa su ejecución después de esta llamada, independientemente de si la indexación ha terminado o no. Si omite este parámetro, la reconstrucción de los índices bloqueará la ejecución del método hasta que se complete la operación de reconstrucción.*

*El comando **RESUME INDEXES** sólo se puede llamar desde 4D Server o un 4D local. Si este comando se ejecuta desde un equipo 4D remoto, se genera el error -10513. Este error puede ser interceptado utilizando un método instalado por el comando **ON ERR CALL**.*

SET EXTERNAL DATA PATH

SET EXTERNAL DATA PATH (aCampo ; ruta)

Parámetro	Tipo		Descripción
aCampo	Texto, BLOB, Imagen, Objeto	➡	Campo para el cual definir el lugar de almacenamiento
ruta	Texto, Entero largo	➡	Ruta de acceso y nombre del archivo de almacenamiento externo o 0 = utilizar la definición en estructura 1 = utilizar la carpeta por defecto

Descripción

*El comando **SET EXTERNAL DATA PATH** define o modifica, para el registro actual, la ubicación de almacenamiento externo del campo aCampo pasado como parámetro.*

4D autoriza almacenar datos de campos de tipo texto, BLOB, Imagen y Objeto fuera del archivo de datos. Para una descripción completa de esta funcionalidad, consulte el Manual de Diseño.

La configuración definida por este comando sólo se aplica cuando el registro actual se guarda en el disco. Los parámetros de almacenamiento definidos en la estructura de la aplicación no cambian. Si el registro actual se cancela, el comando no hace nada. Los parámetros de almacenamiento definidos en la estructura de la aplicación no se modifican.

En ruta, puede pasar una ruta de acceso personalizada o una constante designando un lugar automático:

- ***ruta de acceso personalizada al archivo***
  En este caso, utilice el almacenamiento externo en "modo personalizado". En este modo ciertas funciones de base de datos 4D no están disponibles automáticamente (ver el Manual de Diseño). En particular, debe administrar usted mismo la creación o modificación de los archivos.
  Puede pasar una ruta relativa al archivo de datos o un ruta absoluta, que debe incluir el nombre y la extensión del archivo de almacenamiento. Debe utilizar la sintaxis del sistema. Para definir una ruta relativa, pase "../" (en Windows) o ":.." (OS X) al inicio de la cadena. Puede designar toda carpeta, incluyendo la carpeta por defecto de los archivos externos de la base (nomBase.ExternalData), en este caso, estos archivos se incluyen en el backup de la base. El archivo designado por el parámetro ruta debe existir y ser accesible en el momento de la ejecución del comando. Tenga en cuenta que si la ruta no es válida (archivo o carpeta inexistentes), se devuelve un error sólo en los casos en que se definió la ruta como absoluta. Cuando se especifica una ruta relativa, debe asegurar su validez ya que no se genera un error si no se encuentra el archivo. Si guarda el archivo externo en la misma carpeta que el archivo de datos o una de sus subcarpetas, 4D considera que la ruta especificada es relativa al archivo de datos y mantiene la relación, incluso cuando se mueve o se cambia el nombre de la carpeta de archivos de datos.
  Tenga en cuenta que esto significa que es posible "compartir" el mismo archivo externo entre varios registros. Todo cambio realizado en este archivo externo está disponibles en todos los registros. En este caso, si varios procesos pueden escribir los mismos campos al mismo tiempo, hay que tener cuidado para evitar accesos concurrentes a través de semáforos, con el fin de no afectar los archivos externos.
- ***ubicación automática***
  *Puede designar dos lugares automáticamente con las siguientes constantes, que se encuentra en el tema **Mantenimiento archivo de datos**:*

Constante	Tipo	Valor	Comentario
Use default folder	Entero largo	1	Los datos pasado en parámetro se almacenarán en la carpeta por defecto, llamada nomBase.ExternalData y ubicada al lado del archivo de datos. En este modo, los datos externos son generados por 4D como si estuvieran al interior del archivo de datos.
Use structure definition	Entero largo	0	4D utilizará los parámetros definidos en la estructura para el almacenamiento del campo (ver manual Modo Diseño). Si pasa de un almacenamiento externo a un almacenamiento interno, el archivo externo no se elimina.

Una vez que se ejecuta este comando, 4D mantiene automáticamente el enlace entre el campo del registro y el archivo en el disco. No es necesario para ejecutar el comando de nuevo (excepto si necesita cambiar la ruta). Si 4D ya no puede acceder a los datos del campo (archivo de almacenamiento renombrado o eliminado, ruta modificada, etc.), el campo está vacío, pero no se genera ningún error.

***Nota:** el comando **SET EXTERNAL DATA PATH** sólo se puede ejecutar en 4D local o 4D Server. No hace nada cuando se ejecuta en un 4D remoto.*

Ejemplo

Usted desea guardar el contenido de un archivo existente en el campo imagen, almacenado fura de los datos, en la carpeta de la base:

```
CREATE RECORD([Photos])
[Photos]Name:="Paris.png"
SET EXTERNAL DATA PATH([Photos]Thumbnail;Get 4D folder(Database folder)+"custom"+Folder separator+[Photos]Name)
 //"/custom/Paris.png" debe existir junto al archivo de estructura
SAVE RECORD([Photos])
```

SET INDEX

SET INDEX (unCampo ; index {; modo} {; *})

Parámetro	Tipo		Descripción
unCampo	Campo	➡	Campo del cual crear o borrar el índice
index	Booleano, Entero	➡	• True=Crear el índice, False=Borrar el índice, o • Crear un índice de tipo: -1=palabras claves, 0=por defecto, 1=B-Tree estándar, 3=B-Tree cluster
modo	Entero largo	➡	Obsoleto (parámetro ignorado)
*		➡	Si se pasa * indexación asincrónica

Descripción

***Nota de compatibilidad**: este comando se conserva por razones de compatibilidad únicamente. Ahora recomendamos el uso de los comandos **CREATE INDEX** y **DELETE INDEX** para la gestión de índices por programación.*

*El comando **SET INDEX** acepta dos sintaxis:*

- *Si pasa un booleano en el parámetro index, el comando crea o borra el índice de campo.*
- *Si pasa un entero el parámetro index, el comando crea un índice del tipo especificado.*

index = Booleano

Para indexar el campo, pase True en index. El comando crea un índice del tipo por defecto. Si el índice ya existe, el comando no hace nada.
Si pasa False en index, el comando borrará todos los índices estándar (es decir, no compuestos ni palabras claves) asociados al campo. Si el índice no existe, el comando no hace nada.

index = Entero

*En este caso, el comando crea un índice del tipo especificado por campo. Puede pasar una de las siguientes constantes, que se encuentran en el tema "**Tipo de índice**":*

Constante	Tipo	Valor	Comentario
Cluster BTree Index	Entero largo	3	Índice de tipo B-Tree utilizando clusters. Este tipo de índice se optimiza cuando el índice contiene pocas palabras claves, es decir cuando los mismos valores se presentan con frecuencia en los datos.
Default Index Type	Entero largo	0	4D define el tipo de índice (excepto los índices de palabras claves) que es el más optimizado en función del contenido del campo.
Keywords Index	Entero largo	-1	Permite la indexación palabra por palabra del contenido del campo. Este tipo de índice sólo puede utilizarse con campos de tipo Texto o Alfa.
Standard BTree Index	Entero largo	1	Índice de tipo B-Tree clásico. Este tipo de índice multi propósito se utiliza en las versiones anteriores de 4D

***Nota:** un índice B-Tree asociado a un campo de tipo texto almacena como máximo los primeros 1024 caracteres del campo. Por lo tanto en este contexto, las búsquedas en las cadenas que contengan más de 1024 caracteres fallarán.*

***SET INDEX** no indexará registros bloqueados; el comando esperará a que el registro sea desbloqueado.*

A partir de la versión 11, el parámetro modo no sirve y si se pasa es ignorado.

*El parámetro opcional * indica una indexación asincrónica (simultánea). Una indexación asincrónica permite al método llamante continuar su ejecución inmediatamente después de la llamada, bien sea que la indexación haya terminado o no. Sin embargo, la ejecución se detendrá si un comando requiere la indexación.*

Notas:

- *Los índices creados por este comando no tienen nombres. No pueden ser borrados por el comando **DELETE INDEX** utilizando la sintaxis basada en el nombre.*
- *Este comando no permite crear o borrar índices compuestos.*
- *Este comando no permite borrar un índice de palabras claves creado por el comando **CREATE INDEX**.*

Ejemplo 1

El siguiente ejemplo indexa el campo [Clientes]ID:

```
UNLOAD RECORD([Clientes])
SET INDEX([Clientes]ID;True)
```

Ejemplo 2

Usted quiere indexar el campo [Clientes]Nombre en modo asincrónico:

```
SET INDEX([Clientes]Nombre;True;*)
```

Ejemplo 3

Creación de un índice de palabras claves:

```
SET INDEX([Libros]Summary;Keywords Index)
```

Table

Table (numTabla | unPtr) -> Resultado

Parámetro	Tipo		Descripción
numTabla | unPtr	Entero largo, Puntero	➡	Número de tabla o Puntero de tabla, o Puntero de campo
Resultado	Entero largo, Puntero	➲	Puntero de tabla, si se pasa un número de tabla Número de tabla, si se pasa un puntero de tabla Número de tabla, si se pasa un puntero de campo

Descripción

*El comando **Table** tiene tres sintaxis diferentes:*

- *Si pasa un número de tabla en numTabla, **Table** devuelve un puntero para la tabla.*
- *Si pasa un puntero de tabla en unPtr, **Table** devuelve el número de la tabla.*
- *Si pasa un puntero de campo en unPtr, **Table** devuelve el número de tabla del campo.*

Ejemplo 1

En este ejemplo, la variable ptrTabla recibe un puntero de la tabla 3 de la base:

```
ptrTabla:=Table(3)
```

Ejemplo 2

*Si pasa ptrTabla (un puntero a la tabla 3) a **Table** devuelve 3. En la siguiente línea, la variable numTabla toma el valor 3:*

```
numTabla:=Table(ptrTabla)
```

Ejemplo 3

En este ejemplo, la variable numTabla es igual al número de la tabla [Tabla3]:

```
numTabla:=Table(->[Tabla3])
```

Ejemplo 4

Este ejemplo, la variable numTabla es igual al número de la tabla a la cual pertenece el campo [Tabla3]Campo1:

```
numTabla:=Table(->[Tabla3]Campo1)
```

Table name

Table name (numTabla | ptrTabla) -> Resultado

Parámetro	Tipo		Descripción
numTabla | *ptrTabla*	*Entero largo, Puntero*	➡	*Número de tabla o puntero de tabla*
Resultado	*Cadena*	⮌	*Nombre de la tabla*

Descripción

*El comando **Table name** devuelve el nombre de la tabla cuyo número o puntero se pasa en numTabla o ptrTabla.*

Ejemplo

*El siguiente es un ejemplo de un método genérico que muestra los registros de una tabla. La referencia a la tabla se pasa como un puntero para la tabla. El comando **Table name** se utiliza para incluir el nombre de la tabla en la barra de títulos de la ventana:*

```
` Método de proyecto SHOW CURRENT SELECTION
` SHOW CURRENT SELECTION ( Puntero )
` SHOW CURRENT SELECTION (->[Tabla])

SET WINDOW TITLE("Registros para "+Table name($1)) ` Fijar el título de la ventana
DISPLAY SELECTION($1->) ` Mostrar la selección
```

Eventos de formulario

- *Activated*
- *After*
- *Before*
- *CALL FORM*
- *CALL SUBFORM CONTAINER*
- *Clickcount*
- *Contextual click*
- *Deactivated*
- *EXECUTE METHOD IN SUBFORM*
- *Form event*
- *In break*
- *In footer*
- *In header*
- *Is waiting mouse up*
- *Outside call*
- *POST OUTSIDE CALL*
- *Right click*
- *SET TIMER*
- *_o_During*

Activated

Activated -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		Devuelve TRUE si el ciclo de ejecución está en activación

Descripción

*El comando **Activated** (obsoleto) devuelve **True** en un método formulario cuando la ventana que contiene el formulario se convierte en la ventana del primer plano del proceso del primer plano.*

***Nota:** este comando es equivalente a usar **Evento formulario** y probar si devuelve el evento On Activate.*

***ATENCIÓN:** no ubique un comando como **TRACE** o **ALERT**en la fase **Activated** del formulario, ya que esto produce un bucle infinito.*

***Nota:** para que el ciclo de ejecución **Activated** se genere, asegúrese de que la propiedad del evento On Activate del formulario se haya seleccionado en el entorno Diseño.*

After

After -> Resultado

Parámetro	***Tipo***	***Descripción***
Resultado	*Booleano*	*Devuelve True si el ciclo de ejecución es un after*

Descripción

After *devuelve True para el ciclo de ejecución After.*

Para que el ciclo de ejecución ***After*** *se genere, asegúrese de que la propiedad del evento On Validate ara el formulario y/o los objetos se haya seleccionado en el entorno Diseño.*

Nota: *este comando es equivalente a usar* ***Evento formulario*** *y probar si devuelve el evento On Validate.*

Before

Before -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		Devuelve True si el ciclo de ejecución es un Before

Descripción

Before *devuelve True para el ciclo de ejecución Before.*

Para que el ciclo de ejecución ***Before*** *se genere, asegúrese de que la propiedad de evento On Load para el formulario y/o los objetos se haya seleccionado en el entorno Diseño.*

Nota: *este comando es equivalente a usar* ***Evento formulario*** *y probar si devuelve el evento On Load.*

CALL FORM

CALL FORM (ventana ; metodo {; param}{; param2 ; ... ; paramN})

Parámetro	Tipo		Descripción
ventana	WinRef	➡	Número de referencia de la ventana
metodo	Texto	➡	Nombre del método proyecto a llamar
param	Expresión	➡	Parámetros pasados al método

Descripción

El comando **CALL FORM** ejecuta el método de proyecto cuyo nombre pasó en metodo con uno o varios param(s) en el contexto de un formulario que se muestra en una ventana, independientemente del proceso al que pertenece la ventana.

Al igual que en la comunicación entre procesos basada en los workers (ver **Sobre workers**), un cuadro de mensaje está asociado a la ventana y se puede utilizar cuando la ventana muestra un formulario (después del evento On Load). **CALL FORM** encapsula el nombre del método y sus argumentos en un mensaje que se ha enviado en el cuadro de mensaje de la ventana. El formulario a continuación, ejecuta el mensaje en su propio proceso. El proceso de llamada puede ser cooperativo o apropiativo, por tanto, esta funcionalidad permite a un proceso apropiativo intercambiar información con los formularios.

En ventana, pase el número de referencia de la ventana que muestra el formulario llamado.

En metodo, pase el nombre del método de proyecto que se ejecutará en el contexto del proceso padre de la ventana.

También puede pasar parámetros al método que utilizan uno o más parámetros param. Pase los parámetros de la misma manera que los pasaría a una subrutina (ver la sección **Pasar parámetros a los métodos**). Al iniciar la ejecución en el contexto del formulario, el método recibe los valores parámetro en $1, $2, etc. Recuerde que los arrays no pueden ser pasados como parámetros a un método. Además, en el contexto del comando **CALL FORM**, las siguientes consideraciones adicionales deben tenerse en cuenta:

- Se permiten punteros a tablas o campos.
- Los punteros a las variables, particularmente las variables locales y de proceso, no se recomiendan ya que estas variables pueden no estar definidas en el momento en que el método de proceso acceda.
- Si pasa un parámetro de tipo Objeto o Colección, 4D crea una copia del objeto o de la colección en el proceso de destino (en lugar de una referencia) si el formulario está en un proceso diferente del que llama al comando **CALL FORM**.

Ejemplo 1

Usted quiere abrir dos ventanas de diálogo diferentes en el mismo formulario, pero con diferentes colores de fondo y diferentes mensajes. También desea enviar mensajes después y mostrarlos en cada ventana de diálogo.

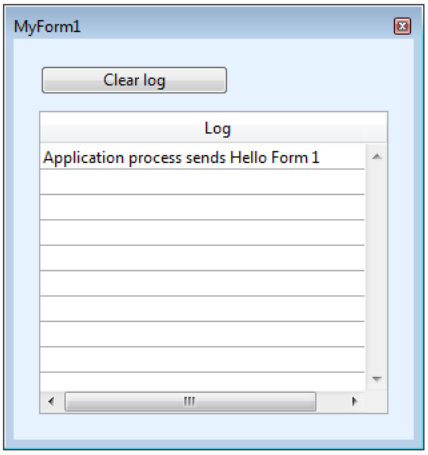
En el formulario principal, un botón abre los dos diálogos:

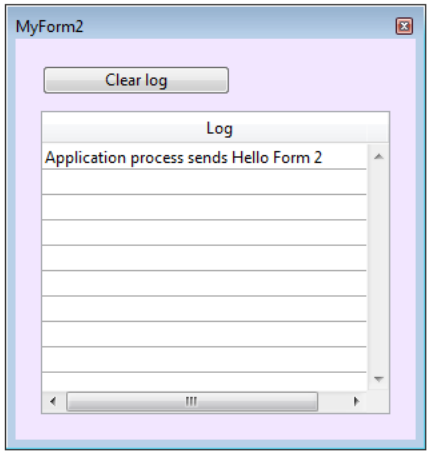
```
 //Crear método objeto formularios
 // primera ventana
formRef1:=Open form window("FormMessage";Palette form window;On the left)
SET WINDOW TITLE("MyForm1";formRef1)
DIALOG("FormMessage";*)
SHOW WINDOW(formRef1)

 //segunda ventana
formRef2:=Open form window("FormMessage";Palette form window;On the left+500)
SET WINDOW TITLE("MyForm2";formRef2)
DIALOG("FormMessage";*)
SHOW WINDOW(formRef2)

 //Enviar colores
CALL FORM(formRef1;"doSetColor";0x00E6F2FF)
CALL FORM(formRef2;"doSetColor";0x00F2E6FF)
 //Dibujar mensajes
CALL FORM(formRef1;"doAddMessage";Current process name;"Hello Form 1")
CALL FORM(formRef2;"doAddMessage";Current process name;"Hello Form 2")
```

El método doAddMessage sólo añade una línea en un list box en el formulario "FormMessage":

```
C_TEXT($1) // nombre del llamador
C_TEXT($2) // mensaje a mostrar
 // recibir mensaje desde $2 y log el mensaje en list box
$p:=OBJECT Get pointer(Object named;"Column1")
INSERT IN ARRAY($p->;1)
$p->{1}:=$1+" sends "+$2
```

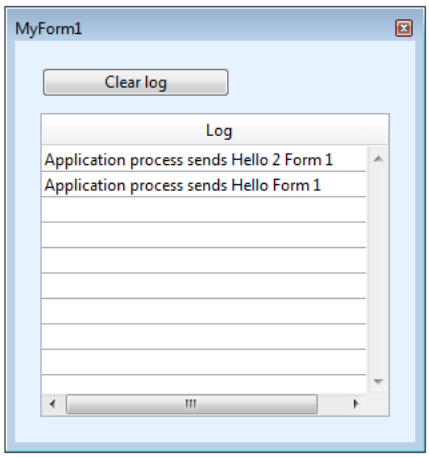
En ejecución, obtiene el siguiente resultado:

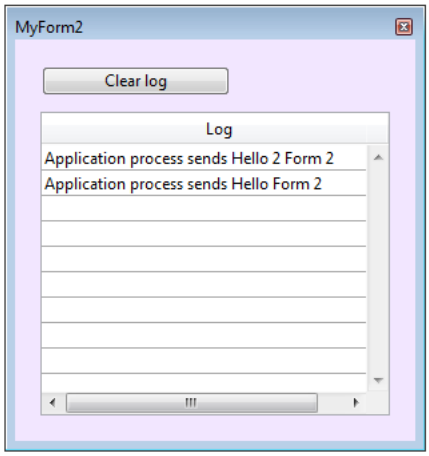




*Luego puede añadir otros mensajes ejecutando el comando **CALL FORM** nuevamente:*

```
CALL FORM(formRef1;"doAddMessage";Current process name;"Hello 2 Form 1")
CALL FORM(formRef2;"doAddMessage";Current process name;"Hello 2 Form 2")
```





Ejemplo 2

*Puede utilizar el comando **CALL FORM** para pasar configuraciones personalizadas a un formulario, por ejemplo valores de configuración, sin tener que utilizar variables proceso:*

```
$win:=Open form window("form")
CALL FORM($win;"configure";param1;param2)
DIALOG("form")
```

CALL SUBFORM CONTAINER

CALL SUBFORM CONTAINER (evento)

Parámetro	Tipo		Descripción
evento	Entero largo	➡	Evento a enviar

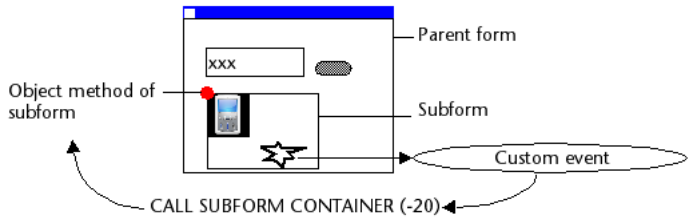
Descripción

*El comando **CALL SUBFORM CONTAINER** permite a una instancia de subformulario enviar el evento al objeto subformulario que lo contiene. El objeto subformulario puede entonces procesar el evento en el contexto del formulario padre.*

Este comando debe ubicarse en el método de formulario del subformulario o en el método de objeto de uno de los objetos de subformulario. El evento sólo se recibirá en el método de objeto del contenedor del subformulario.

*En evento, puede pasar todo evento de formulario predefinido de 4D (puede utilizar las constantes del tema "**Eventos de formulario**") o todo valor correspondiente a un evento personalizado. En el caso de un evento personalizado, se recomienda pasar un valor negativo en evento para evitar el riesgo de interferir con los números de eventos existentes o futuros de 4D.*

*Ejemplo de ejecución del comando **CALL SUBFORM CONTAINER**:*



Clickcount

Clickcount -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo	➲	Número de clics consecutivos

Descripción

*El comando **Clickcount** devuelve, en el contexto de un evento clic, el número de veces que el usuario ha hecho clic de manera repetida con el mismo botón del ratón. Normalmente, este comando devuelve 2 para un doble clic.*

Este comando le permite detectar un doble clic en los encabezados o pies de list box e igualmente manejar las secuencias de triples clics o más.

*Cada clic con un botón del ratón genera un evento clic separado. Por ejemplo, si un usuario hace doble clic, un evento es generado para el primer clic en el cual **Clickcount** devuelve 1; luego otro evento es generado por el segundo clic, en el cual **Clickcount** devuelve 2.*

Este comando sólo debe ser utilizado en el contexto del evento formulario On Clicked, On Header Click u On Footer Click. Por tanto, es necesario verificar en modo Diseño que el evento correspondiente ha sido seleccionado correctamente en las propiedades del formulario y/o para el objeto específico.

*Cuando ambos eventos formulario On Clicked y On Double Clicked están activados, la siguiente secuencia será devuelto por **Clickcount**:*

- *1 en el evento On Clicked*
- *2 en el evento On Double Clicked*
- *2+n en el evento On Clicked*

Ejemplo 1

La estructura de código siguiente se puede colocar en un encabezado de listbox para manejar clics simples y dobles:

```
Case of
   :(Form event=On Header Click)
      Case of
         :(Clickcount=1)
            ... //single-click action
         :(Clickcount=2)
            ... //double-click action
      End case
End case
```

Ejemplo 2

Las etiquetas no son editables pero lo son después de un triple-clic. Si desea permitir a los usuarios editar las etiquetas, puede escribir el método objeto siguiente:

```
If(Form event=On Clicked)
   Case of
      :(Clickcount=3)
         OBJECT SET ENTERABLE(*;"Label";True)
         EDIT ITEM(*;"Label")
   End case
End if
```

Contextual click

Contextual click -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		Verdadero si se detecta un clic contextual, de lo contrario Falso

Descripción

*El comando **Contextual click** devuelve Verdadero si un se ha efectuado un clic contextual:*

- *Bajo Windows y Mac OS, los clics contextuales se efectúan con el botón derecho del ratón.*
- *Bajo Mac OS, los clics contextuales también pueden generarse utilizando la combinación **Control+clic**.*

*Este comando debe utilizarse sólo en el contexto del evento de formulario **On clicked**. Por lo tanto es necesario verificar en modo Diseño que el evento haya sido seleccionado correctamente en las propiedades del formulario y/o del objeto específico.*

Ejemplo

Este método, combinado con un área desplegable, le permite cambiar el valor de un elemento array utilizando un menú contextual:

```
If (Contextual click)
    If (Pop up menu ("True;False")=1)
        miArray{miArray}:="True"
    Else
        miArray{miArray}:="False"
    End if
End if
```

Deactivated

Deactivated -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		Devuelve TRUE si el ciclo de ejecución está en desactivación

Descripción

*El comando **Deactivated** devuelve True en un método formulario u objeto cuando la ventana del primer plano del proceso de primer plano, contiene el formulario, se traslada a la parte posterior.*
*Para que se genere el ciclo de ejecución **Deactivated**, asegúrese de que la propiedad del evento On Deactivate para el formulario y/o los objetos haya sido seleccionada en el entorno Diseño.*

***Nota:** este comando es equivalente a utilizar **Evento formulario** y probar si devuelve el evento On Deactivate.*

EXECUTE METHOD IN SUBFORM

EXECUTE METHOD IN SUBFORM (objetoSubForm ; nomMetodo {; retorno {; param} {; param2 ; ... ; paramN}})

Parámetro	Tipo		Descripción
objetoSubForm	Texto	➡	Nombre del objeto subformulario
nomMetodo	Texto	➡	Nombre del método proyecto a ejecutar
retorno	Operador, Variable	➡	* si el método no devuelve un valor
		⬅	Valor devuelto por el método
param	Expresión	➡	Parámetro(s)a pasar al método

Descripción

*El comando **EXECUTE METHOD IN SUBFORM** permite ejecutar el método proyecto nomMetodo en el contexto del subformulario objetoSubform.*

*El método de proyecto llamado puede recibir llamadas 1 a X parámetros en param y devolver un valor en retorno. Pase * en el parámetro retorno si el método no devuelve parámetros.*

Puede pasar en nomMetodo el nombre de todo método de proyecto accesible desde la base o el componente que ejecuta el comando. El contexto de ejecución se conserva en el método llamado, lo que significa que el formulario actual y el evento formulario actual siguen definidos. Si el subformulario proviene de un componente, el método debe pertenecer al componente y tener la propiedad "Compartido entre los componentes y la base local".

Este comando debe llamarse en el contexto del formulario padre (contiene el objeto objetoSubForm), por ejemplo vía el método de formulario.

***Nota:** el método nomMetodo no se ejecuta si objetoSubForm no se encuentra en la página actual o si no ha sido instanciado.*

Ejemplo 1

*Dado el formulario "ContactDetail" utilizado como subformulario en el formulario padre "Empresa". El objeto subformulario que contiene el formulario ContactDetail se llama "ContactSubform". Imagine que queremos modificar la apariencia de ciertos elementos del subformulario de acuerdo al valor de los campos de la empresa (por ejemplo, "nomcontact" debe pasar a rojo cuando [Empresa]Ciudad="Nueva York" y a azul cuando [Empresa]Ciudad="San Diego"). Este mecanismo se implementa vía el método **SetToColor**. Para poder obtener este resultado, el método **SetToColor** no puede llamarse directamente desde el proceso del evento de formulario "On Load" del formulario padre Empresa porque el objeto "contactname" no pertenece al formulario actual, sino al formulario mostrado en el objeto subformulario "ContactSubform". El método debe por lo tanto ejecutarse utilizando el comando **EXECUTE METHOD IN SUBFORM** para que funcione correctamente.*

```
Case of
   :(Form event=On Load)
      Case of
         :([Empresa]Ciudad ="Nueva York")
            $Color:=$Red
         :([Empresa]Ciudad ="San Diego")
            $Color:=$Blue
         Else
            $Color:=$Black
      End case
      EXECUTE METHOD IN SUBFORM("ContactSubform";"SetToColor";*;$Color)
End case
```

Ejemplo 2

*Desarrolle una base que se utilizará como componente. Incluye un formulario proyecto compartido (llamado por ejemplo Calendar) que contiene variables dinámicas así como también un método proyecto público que permite ajustar el calendario: **SetCalendarDate(varDate)**.*
Si este método se utilizó directamente en el método del formulario Calendar, puede llamar directamente el evento "On Load":

```
SetCalendarDate(Current date)
```

*Pero, en el contexto de la base local, imaginemos que un formulario proyecto contiene los dos subformularios "Calendar", en los objetos subformulario llamados "Cal1" y "Cal2". Debe definir la fecha de Cal1 en 01/01/10 y la fecha de Cal2 en 05/05/10. No puede llamar directamente a **SetCalendarDate** porque el método no "sabrá" que formularios y variables aplicar. Por lo tanto, debe llamarlo vía el siguiente código:*

```
EXECUTE METHOD IN SUBFORM("Cal1";"SetCalendarDate";*;!01/01/10!)
EXECUTE METHOD IN SUBFORM("Cal2";"SetCalendarDate";*;!05/05/10!)
```

Ejemplo 3

Ejemplo avanzado: en el mismo contexto anterior, este ejemplo ofrece un método genérico:

```
  // Contenido del método SetCalendarDate
C_DATE($1)
C_TEXT($2)
Case of
    :(Count parameters=1)
  // Ejecución estándar del método (como si se hubiera
  // ejecutado desde el formularios mismo)
        o
  //especificamente para un contexto (ver caso 2)

    :(Count parameters=2)
  // Llamada externa, necesita un contexto
  // Llamada recursiva con un solo parámetro
        EXECUTE METHOD IN SUBFORM($2;"SetCalendarDate";*;$1)
End case
```

Variables y conjuntos del sistema

Si este comando se ejecuta correctamente, la variable sistema OK toma el valor 1; de lo contrario toma el valor 0.

Form event

Form event -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entero largo*	➲	*Número del evento de formulario*

Descripción

Evento formulario *devuelve un valor numérico que identifica el tipo de evento de formulario que acaba de ocurrir. Generalmente,* ***Evento formulario*** *se utiliza en un método formulario o en un método objeto.*

4D ofrece constantes predefinidas (ubicadas en el tema ***Eventos de formulario****) para comparar los valores devueltos por el comando* ***Evento formulario*** *.*

Ciertos eventos son genéricos (generados por todo tipo de objeto) y otros son específicos a un tipo de objeto particular.

Constante	Tipo	Valor	Comentario
On Load	Entero largo	1	El formulario se muestra o se imprime
On Mouse Up	Entero largo	2	(Sólo imágenes) El usuario acaba de liberar el botón izquierdo del ratón en un objeto Imagen
On Validate	Entero largo	3	Se ha valido la entrada de datos en el registro
On Clicked	Entero largo	4	Ocurre un clic sobre un objeto
On Header	Entero largo	5	El encabezado del formulario se va a imprimir o a mostrar
On Printing Break	Entero largo	6	Se va a imprimir una de las áreas de ruptura del formulario
On Printing Footer	Entero largo	7	Se va a imprimir el área de pie de página del formulario
On Display Detail	Entero largo	8	Un registro se va a mostrar en una lista o una línea se va a mostrar en un list box.
On VP Ready	Entero largo	9	(Áreas 4D View Pro únicamente) La carga del área 4D View Pro está completa
On Outside Call	Entero largo	10	El formulario recibe una llamada **POST OUTSIDE CALL**
On Activate	Entero largo	11	La ventana del formulario se convierte en la ventana del primer plano
On Deactivate	Entero largo	12	La ventana del formulario deja de ser la ventana del primer plano
On Double Clicked	Entero largo	13	Un objeto ha recibido un doble clic
On Losing Focus	Entero largo	14	Un objeto de formulario está perdiendo el foco
On Getting Focus	Entero largo	15	Un objeto de formulario toma el foco
On Drop	Entero largo	16	Se han soltado datos en un objeto
On Before Keystroke	Entero largo	17	Un carácter está a punto de introducirse en el objeto que tiene el foco. **Get edited text** devuelve el texto del objeto sin este carácter.
On Menu Selected	Entero largo	18	Se ha seleccionado un comando de menú
On Plug in Area	Entero largo	19	Un objeto externo solicitó que se ejecute su método de objeto
On Data Change	Entero largo	20	Se han modificado los datos de un objeto
On Drag Over	Entero largo	21	Los datos pueden soltarse en un objeto
On Close Box	Entero largo	22	Se ha hecho clic en la casilla de cerrar la ventana
On Printing Detail	Entero largo	23	Se va a imprimir el área de detalle del formulario
On Unload	Entero largo	24	El formulario se cierra y libera
On Open Detail	Entero largo	25	El formulario detallado asociado con el formulario de salida o con el listbox está apunto de ser abierto
On Close Detail	Entero largo	26	Se cierra el formulario de entrada y regresa al formulario de salida
On Timer	Entero largo	27	El número de tics definido por el comando **SET TIMER** se ha pasado
On After Keystroke	Entero largo	28	Un carácter está apunto de introducirse en el objeto que tiene el foco. **Get edited text** devuelve el contenido incluyendo este carácter
On Resize	Entero largo	29	La ventana del formulario se redimensiona
On After Sort	Entero largo	30	(List box únicamente) Se acaba de efectuar una ordenación estándar en una columna del list box
On Selection Change	Entero largo	31	• List box: se modifica la selección actual de líneas o columnas • Registros en lista: se modifica el registro actual o la selección actual de líneas en un formulario listado o en un subformulario • Lista jerárquica: la selección en la lista se modifica luego de un clic o de presionar una tecla • Variable o campo editable: la selección de texto o la posición del cursor en el área se modifica al hacer clic o presionar una tecla.
On Column Moved	Entero largo	32	(List box únicamente) El usuario mueve una columna de list box vía arrastrar y soltar
On Column Resize	Entero largo	33	(List box únicamente) El ancho de una columna de list box es modificado por un usuario con el ratón

Constante	Tipo	Valor	Comentario
On Row Moved	Entero largo	34	(List box únicamente) El usuario mueve una fila de un list box vía arrastrar y soltar
On Mouse Enter	Entero largo	35	El cursor del ratón entra al área gráfica de un objeto
On Mouse Leave	Entero largo	36	El cursor del ratón sale del área gráfica de un objeto
On Mouse Move	Entero largo	37	El cursor del ratón se mueve al menos un píxel O cuando se presiona una tecla de modificación (Ctrl, Alt, Bloq mayús). Si el evento está seleccionado para un objeto únicamente, se genera sólo cuando el cursor se encuentra dentro del área gráfica del objeto
On Alternative Click	Entero largo	38	• Botones 3D: el área "flecha" de un botón 3D recibe un clic • List boxes: en una columna de un array objeto, un botón de puntos suspensivos (atributo "alternateButton") recibe un clic **Nota**: los botones de elipses están disponibles sólo para versiones v15 o superiores.
On Long Click	Entero largo	39	(Botones 3D únicamente) Se hace clic en un botón 3D y el botón del ratón permanece presionado por un cierto tiempo
On Load Record	Entero largo	40	En modo entrada en lista, se carga un registro durante modificación (el usuario hace clic en una línea del registro y un campo pasa a modo edición)
On Before Data Entry	Entero largo	41	(List box únicamente) Una celda de list box está a punto de pasar a modo edición
On Header Click	Entero largo	42	(List box únicamente) Ocurre un clic en un encabezado de columna del list box
On Expand	Entero largo	43	(Listas jerárquicas únicamente) Se ha expandido un elemento de la lista jerárquica utilizando un clic o una tecla
On Collapse	Entero largo	44	(Listas jerárquicas únicamente) Un elemento de la lista jerárquica se ha contraído vía un clic o una tecla
On After Edit	Entero largo	45	El contenido del objeto editable que tiene el foco acaba de ser modificado
On Begin Drag Over	Entero largo	46	Se va a arrastrar un objeto
On Begin URL Loading	Entero largo	47	(Áreas web únicamente) Un nuevo URL se carga en el área web
On URL Resource Loading	Entero largo	48	(Áreas web únicamente) Se carga un nuevo recurso en el área web
On End URL Loading	Entero largo	49	(Áreas web únicamente) Se han cargado todos los recursos del URL
On URL Loading Error	Entero largo	50	(Áreas web únicamente) Ocurrió un error cuando se cargaba el URL
On URL Filtering	Entero largo	51	(Áreas web únicamente) Un URL fue bloqueado por el área web
On Open External Link	Entero largo	52	(Áreas web únicamente) Se ha abierto un URL externo en el navegador
On Window Opening Denied	Entero largo	53	(Áreas web únicamente) Se ha bloqueado una ventana pop-up
On bound variable change	Entero largo	54	Se modifica la variable relacionada a un subformulario.
On Page Change	Entero largo	56	Al cambiar de página actual en el formulario
On Footer Click	Entero largo	57	(List box únicamente) Un clic en el pie de un list box o de una columna de list box
On Delete Action	Entero largo	58	(Listas jerárquicas y list box únicamente). El usuario solicita borrar un elemento.
On Scroll	Entero largo	59	El usuario desplaza el contenido de un campo o de una variable imagen utilzando el ratón o una tecla.

Nota: *los eventos específicos de formularios de salida no pueden utilizarse en* ***formularios proyecto****. Estos son: On Display Detail, On Open Detail, On Close Detail, On Load Record, On Header, On Printing Detail, On Printing Break, On Printing Footer.*

Eventos y métodos

Cuando ocurre un evento de formulario, 4D efectúa las siguientes acciones:

- *Primero, examina los objetos del formulario y llama al método de objeto para todos los objetos (involucrados en el evento) cuya propiedad de evento de objeto correspondiente ha sido seleccionada.*
- *Segundo, llama al método de formulario si la propiedad de evento de formulario correspondiente ha sido seleccionada.*

No asuma que los métodos de objeto, si los hay, se llamarán en un orden particular. La regla es que los métodos de objeto siempre se llaman antes que los métodos de formulario. Si un objeto es un subformulario, se llama primero a los métodos de objeto del formulario de salida del subformulario, luego el método de formulario del formulario de salida. 4D luego continúa llamando a los métodos del formulario padre. En otras palabras, cuando un objeto está en un subformulario, 4D utiliza la misma regla que para los métodos de objeto y formulario en subformularios.

Excepto para los eventos On Load y On Unload, si la propiedad de evento de formulario no está seleccionada para un evento dado, esto no evita las llamadas a los métodos de objeto para los objetos cuya propiedad de evento está seleccionada. Es decir, la activación o desactivación de un evento a nivel del formulario no tiene efecto en las propiedades del evento de los objetos.

El número de objetos involucrados en un evento depende de la naturaleza del evento:

- *Evento On Load, los métodos de objeto de todos los objetos del formulario (de todas las páginas) que tengan seleccionada la propiedad de evento On Load serán llamados. Luego, si la propiedad de evento de formulario On Load está seleccionada, el método de formulario será llamado.*
- *Eventos On Activate u On Resize, ningún método de objeto será llamado, porque este evento aplica al formulario como un todo y no como un objeto en particular. Por lo tanto, si el evento de formulario On Activate está seleccionado, sólo se llamará al método de formulario.*
- *Evento On Timer, este evento se genera únicamente si el método de formulario contiene una llamada previa al comando **SET TIMER**. Si la propiedad de evento de formulario On Timer está seleccionada, sólo el método de formulario recibirá el evento, no se llamará al método de formulario.*
- *Evento On Drag Over, sólo se llamará el método del objeto soltable involucrado en el evento (si la propiedad de evento "Soltable" está seleccionada para el objeto). No se llamará el método de formulario.*
- *Por el contrario, para el evento On Begin Drag over, se llamará el método del objeto o el método del formulario del objeto arrastrado (Si la propiedad de evento "Arrastrable" está seleccionada para el objeto).*

***Advertencia:** a diferencia de otros eventos, durante un evento On Begin Drag over, el método llamado se ejecuta en el contexto del proceso de arrastrar y soltar del objeto fuente, no en el del proceso de arrastrar y soltar el objeto de destino. Para mayor información, consulte la sección **Arrastrar y soltar**.*

- *Si los eventos On Mouse Enter, On Mouse Move y On Mouse Leave han sido seleccionados para el formulario, se generan para cada objeto del formulario. Si están seleccionados para un objeto, son generados sólo para el objeto. En caso de superposición de objetos, el evento se genera por el primer objeto capaz de administrarlo que se encuentre del nivel superior al inferior. Los objetos que se hicieron invisibles utilizando el comando **OBJECT SET VISIBLE** no generan estos eventos. Durante la entrada de datos, otros objetos pueden recibir este tipo de eventos dependiendo de la posición del ratón.*
  *Note que el evento On Mouse Move se genera cuando el cursor del ratón se mueve pero también cuando el usuario presiona una tecla de modificación como **Mayús**, **Bloq Mayús**, **Ctrl** u **Opción** (esto permite manejar las operaciones de arrastrar y soltar de tipo copia o desplazamiento).*
- *Registros en lista: la secuencia de llamadas a métodos y eventos de formularios en formularios listados visualizados a través de **DISPLAY SELECTION** / **MODIFY SELECTION** y los subformularios es la siguiente:*
  Para cada objeto del área de encabezado:
    - *Método objeto con evento On Header*
    - *Método formulario con evento On Header*
  Para cada registro:
    - *Para cada objeto en el área de detalle:*
      - *Método de objeto con evento On Display Detail*
    - *Método de formulario con evento On Display Detail*

- *No se permite llamar un comando 4D que muestre una caja de diálogo desde los eventos On Display Detail y On Header y provoca un error de sintaxis.*
  *Más particularmente, los comandos relacionados son: **ALERT**, **DIALOG**, **CONFIRM**, **Request**, **ADD RECORD**, **MODIFY RECORD**, **DISPLAY SELECTION** y **MODIFY SELECTION**.*
- *On Page Change: este evento sólo está disponible a nivel de los formularios (se llama en el método formulario). Se genera cada vez que cambia la página actual del formulario (seguido de una llamada al comando **FORM GOTO PAGE** o de una acción estándar de navegación). Note que el evento se genera luego de la carga completa de la página, es decir una vez todos los objetos que contiene se inicializan (incluyendo las áreas web). Este evento es útil para ejecutar código que necesite que todos los objetos se inicialicen de antemano. También puede utilizarlo para optimizar la aplicación al ejecutar código (por ejemplo una búsqueda) sólo después de la visualización de una página específica del formulario y no tan pronto como se carga la página 1. Si el usuario no accede a esta página, el código no se ejecuta.*

La siguiente tabla resume cómo se llaman los métodos de formulario y objetos para cada tipo de evento:

Evento	*Métodos de objeto*	*Método de formulario*	*Qué objetos*
On Load	*Sí*	*Sí*	*Todos*
On Unload	*Sí*	*Sí*	*Todos*
On Validate	*Sí*	*Sí*	*Todos*
On Clicked	*Sí (si cliqueable o editable) (*)*	*Sí*	*Sólo el objeto implicado*
On Double Clicked	*Sí (si cliqueable o editable) (*)*	*Sí*	*Sólo el objeto implicado*
On Before Keystroke	*Sí (si editable) (*)*	*Sí*	*Sólo el objeto implicado*
On After Keystroke	*Sí (si editable) (*)*	*Sí*	*Sólo el objeto implicado*
On After Edit	*Sí (si editable) (*)*	*Sí*	*Sólo el objeto implicado*
On Getting Focus	*Sí (si tabulable) (*)*	*Sí*	*Sólo el objeto implicado*
On Losing Focus	*Sí (si tabulable) (*)*	*Sí*	*Sólo el objeto implicado*
On Activate	*Nunca*	*Sí*	*Ninguno*
On Deactivate	*Nunca*	*Sí*	*Ninguno*
On Outside Call	*Nunca*	*Sí*	*Ninguno*
On Begin drag over	*Sí (si arrastrable) (**)*	*Sí*	*Sólo el objeto implicado*
On Drop	*Sí (si soltable) (**)*	*Sí*	*Sólo el objeto implicado*
On Drag Over	*Sí (si soltable) (**)*	*Nunca*	*Sólo el objeto implicado*
On Mouse Enter	*Sí*	*Sí*	*Todos*
On Mouse Move	*Sí*	*Sí*	*Todos*
On Mouse Leave	*Sí*	*Sí*	*Todos*
On Mouse Up	*Sí*	*Nunca*	*Sólo el objeto implicado*
On Menu Selected	*Nunca*	*Sí*	*Ninguno*
On bound variable change	*Nunca*	*Sí*	*Ninguno*
On Data Change	*Sí (si modificable) (*)*	*Sí*	*Sólo el objeto implicado*
On Plug in Area	*Sí*	*Sí*	*Sólo el objeto implicado*
On Header	*Sí*	*Sí*	*Todos*
On Printing Detail	*Sí*	*Sí*	*Todos*
On Printing Break	*Sí*	*Sí*	*Todos*
On Printing Footer	*Sí*	*Sí*	*Todos*
On Close Box	*Nunca*	*Sí*	*Ninguno*
On Display Detail	*Sí*	*Sí*	*Todos*
On Open Detail	*No, excepto para List boxes*	*Sí*	*Ninguno excepto List box*
On Close Detail	*No, excepto para List boxes*	*Sí*	*Ninguno excepto List box*
On Resize	*Nunca*	*Sí*	*Ninguno*
On Selection Change	*Sí (***)*	*Sí*	*Sólo el objeto implicado*
On Load Record	*Nunca*	*Sí*	*Ninguno*
On Timer	*Nunca*	*Sí*	*Ninguno*
On Scroll	*Sí*	*Nunca*	*Objeto involucrado únicamente*
On Picture Scroll	*Sí*	*Sí*	*Objeto involucrado únicamente*
On Before Data Entry	*Sí (List box)*	*Nunca*	*Sólo el objeto implicado*
On Column Moved	*Sí (List box)*	*Nunca*	*Sólo el objeto implicado*
On Row Moved	*Sí (List box)*	*Nunca*	*Sólo el objeto implicado*
On Column Resize	*Sí (List box)*	*Nunca*	*Sólo el objeto implicado*
On Header Click	*Sí (List box)*	*Nunca*	*Sólo el objeto implicado*
On After Sort	*Sí (List box)*	*Nunca*	*Sólo el objeto implicado*
On Long Click	*Sí (Botón 3D)*	*Sí*	*Sólo el objeto implicado*
On Alternative Click	*Sí (Botón 3D y list box)*	*Nunca*	*Sólo el objeto implicado*
On Expand	*Sí (Lista jerárq.)*	*Nunca*	*Sólo el objeto implicado*
On Collapse	*Sí (Lista jerárq.)*	*Nunca*	*Sólo el objeto implicado*
On Delete Action	*Sí (Lista jerárq. y list box)*	*Nunca*	*Sólo el objeto implicado*
On URL Resource Loading	*Sí (Web Area)*	*Nunca*	*Sólo el objeto implicado*
On Begin URL Loading	*Sí (Web Area)*	*Nunca*	*Sólo el objeto implicado*
On URL Loading Error	*Sí (Web Area)*	*Nunca*	*Sólo el objeto implicado*
On URL Filtering	*Sí (Web Area)*	*Nunca*	*Sólo el objeto implicado*
On End URL Loading	*Sí (Web Area)*	*Nunca*	*Sólo el objeto implicado*
On Open External Link	*Sí (Web Area)*	*Nunca*	*Sólo el objeto implicado*
On Window Opening Denied	*Sí (Web Area)*	*Nunca*	*Sólo el objeto implicado*
On VP Ready	*Sí (Área 4D View Pro)*	*Sí*	*Sólo el objeto implicado*

(*) Para mayor información, ver la sección "Eventos, objetos y propiedades" a continuación.
*(**) Consulte la sección "**Arrastrar y soltar**" para mayor información.*
(***) Sólo los objetos de tipo list box, lista jerárquica y subformulario soportan este evento.

***IMPORTANTE:** siempre tenga en cuenta que, para cualquier evento, el método de un formulario o de un objeto se llama si el evento correspondiente es seleccionado para el formulario u objeto. El beneficio de desactivar eventos en el entorno Diseño (utilizando la Lista de propiedades del editor de formularios) es que usted puede reducir de manera importante el número de llamadas a métodos y por lo tanto optimizar de manera significativa la velocidad de ejecución de sus formularios.*

***Advertencia:** los eventos On Load y On Unload son generados por objetos si están activados para el objeto y el formulario al cual pertenece el objeto. Si los eventos están activados para el objeto únicamente, no ocurrirán; estos dos eventos también deben activarse a nivel del formulario.*

Eventos, objetos y propiedades

Un método de objeto es llamado si el evento puede realmente ocurrir para el objeto, dependiendo de su naturaleza y propiedades. La siguiente sección detalla los eventos que usted utilizará generalmente para manejar los diferentes tipos de objetos.

Recuerde que la Lista de propiedades del editor de formularios sólo muestra los eventos compatibles con el objeto seleccionado o el formulario.

Objetos cliqueables

Los objetos cliqueables son administrables principalmente con el ratón. Son los siguientes:

- *Variables o campos editables de tipo Booleano*
- *Botones, botones por defecto, botones de opción, casillas de selección, rejillas de botones*
- *Botones 3D, Botones de opción 3D, Casillas de selección 3D*
- *Menús desplegables, menús jerárquicos desplegables, menús imagen*
- *Listas desplegables, menús*
- *Áreas de desplazamiento, listas jerárquicas, list boxes y columnas de list box*
- *Botones invisibles, botones inversos, botones opción imagen*
- *Termómetros, reglas, dials (también conocidos como objetos deslizables)*
- *Pestañas*
- *Separadores.*

Cuando el evento On Clicked u On Double Clicked se selecciona para uno de estos objetos, puede detectar y administrar los clics en el objeto, utilizando el comando ***Evento formulario*** *que devuelve On Clicked u On Double Clicked, dependiendo del caso. Si ambos eventos están seleccionados para un objeto, los eventos On Clicked y On Double Clicked serán generados cuando el usuario haga doble clic en el objeto.*

Nota: *a partir de 4D v14, los campos y variables editables que contienen texto (tipo texto, fecha, hora o número) también generan los eventos On Clicked y On Double Clicked.*

Para todos estos objetos, el evento On Clicked ocurre una vez se libera el botón del ratón. Sin embargo, hay varias excepciones:

- *Botones invisibles - El evento On Clicked ocurre tan pronto como se hace clic y no espera a que se libere el botón del mouse.*
- *Objetos deslizables (termómetros, reglas, y dials) - Si el formato de salida indica que el método de objeto debe llamarse mientras usted desliza el control, el evento On Clicked ocurre tan pronto como se hace clic.*

En el contexto del evento On Clicked, puede probar el número de clics efectuados por el usuario con la ayuda del comando ***Clickcount****.*

Nota: *algunos de estos objetos pueden activarse con el teclado. Por ejemplo, una vez que una casilla de selección obtiene el foco, puede seleccionarse utilizando la barra espaciadora. En tal caso, se genera un evento On Clicked.*

Advertencia: *los combo boxes no se consideran objetos cliqueables. Un combo box debe tratarse como un área de texto editable cuya lista desplegable asociada ofrece valores por defecto. Por lo tanto, usted puede manejar entrada de datos en un combo box con la ayuda de los eventos On Before Keystroke, On After Keystroke y On Data Change.*

Nota*: a partir de 4D v13, los objetos de tipo pop up/lista desplegable y menú desplegable jerárquico pueden generar el evento On Data Change. Esto le permite detectar la activación del objeto cuando se selecciona un valor diferente del valor actual.*

Objetos editables por teclado

Los objetos editables por teclado son objetos en los cuales usted introduce datos utilizando el teclado y para los cuales puede filtrar los datos de entrada al menor nivel detectando los eventos On After Edit, On Before Keystroke, On After Keystroke y On Selection Change.

Los objetos y tipos de datos editables son los siguientes:

- *Todos los campos editables de tipo alfa, texto, fecha, hora, numérico u (On After Edit únicamente) imagen*
- *Todas las variables editables de tipo alfa, texto, fecha, hora, numérico u (On After Edit únicamente) imagen*
- *Combo boxes (excepto On Selection Change)*
- *List boxes.*

Nota*: a partir de 4D v14, los campos y variables editables que contienen texto (tipo texto, fecha, hora o número) también generan los eventos On Clicked y On Double Clicked.*

Nota: *aunque son objetos "editables", las listas jerárquicas no manejan los eventos formulario On After Edit, On Before Keystroke y On After Keystroke (Ver también el párrafo "Listas jerárquicas" a continuación).*

- *On Before Keystroke y On After Keystroke*

Nota: *a partir de la versión 2004.2 de 4D, el evento On After Keystroke puede generalmente ser reemplazado por el evento On After Edit (ver a continuación).*

Una vez los eventos On Before Keystroke y On After Keystroke hayan sido seleccionados para un objeto, puede detectar y administrar las pulsaciones de teclas en el objeto, utilizando el comando ***Evento formulario*** *que devolverá On Before Keystroke y luego On After Keystroke (para mayor información, consulte la descripción del comando* ***Get edited text****). Estos eventos también son activados por comandos de lenguaje que simulan la acción del usuario, tales como* ***POST KEY****.*

Recuerde que las modificaciones del usuario que no se llevan a cabo utilizando el teclado (pegar, arrastrar-soltar, etc.) no se tienen en cuenta. Para procesar estos eventos, debe utilizar On After Edit.

Nota: *los eventos On Before Keystroke y On After Keystroke no se generan durante la utilización de un método de entrada. Un método de entrada (o IME, Input Method Editor) es un programa o un componente sistema que puede utilizarse para introducir caracteres complejos o símbolos (por ejemplo, japoneses o chinos) utilizando un teclado occidental.*

- *On After Edit*
  Cuando se utiliza, este evento se genera después de cada cambio realizado al contenido de un objeto editable, sin importar la acción que originó el cambio, es decir:
  - Las acciones de edición estándar que modifican el contenido tales como pegar, cortar, borrar o cancelar;

- Soltar un valor (acción similar a pegar);
- Toda entrada de teclado realizada por el usuario; en este caso, el evento On After Edit se genera después de los eventos On Before Keystroke y On After Keystroke, si se utilizan.
*- Toda modificación realizada utilizando un comando de lenguaje que estimula una acción de usuario (por ejemplo **POST KEY**).*
Atención, las siguientes acciones NO activan este evento:
- Las acciones de edición que no modifican el contenido del área, como copiar o seleccionar todo, o arrastrar un valor (acción similar a copiar); sin embargo, estas acciones modifican la ubicación del cursor y desencadenan el evento On Selection Change.
- Las modificaciones a los contenidos por programación, excepto para los comandos que simulan una acción de usuario.
Este evento puede utilizarse para controlar acciones de usuario con el fin de prevenir que peguen un texto muy largo, bloquear ciertos caracteres o evitar que se corte un campo de contraseña.

- *On Selection Change: cuando se aplica a un campo o variable de texto dinámico (editable o no), este evento se dispara cada vez que la posición del cursor cambia. Esto sucede, por ejemplo, tan pronto como el usuario selecciona el texto utilizando el ratón o las teclas de flecha, o cuando el usuario introduce texto. Esto le permite llamar, por ejemplo, comandos como **GET HIGHLIGHT**.*

Objetos modificables

Los objetos modificables tienen una fuente de datos cuyos valores pueden modificarse utilizando el ratón o el teclado; no son considerados verdaderamente como controles de interfaz de usuario manejados a través del evento On Clicked. Estos objetos son los siguientes:

- *Todos los campos editables (excepto BLOBs)*
- *Todas las variables editables (excepto BLOBs, punteros, y arrays)*
- *Combo boxes*
- *Objetos externos (para los cuales la entrada de datos es validada por el plug-in)*
- *Listas jerárquicas*
- *List boxes y columnas de list box.*

*Estos objetos reciben eventos On Data Change. Cuando el evento On Data Change está seleccionado para uno de estos objetos, usted puede detectar y administrar el cambio de los valores de la fuente de datos, utilizando el comando **Evento formulario** que devolverá On Data Change. El evento es generado tan pronto como la variable asociada con el objeto sea actualizada internamente por 4D (por lo general, cuando el área de entrada del objeto pierde el foco).*

Objetos tabulables

Los objetos tabulables obtienen el foco cuando utiliza la tecla Tab para alcanzarlos y/o cuando hace clic en ellos. El objeto que tiene el foco recibe los caracteres (digitados en el teclado) que no son modificadores de un comando de menú o de un objeto tal como un botón.

Todos los objetos son tabulables, EXCEPTO los siguientes:

- *Variables o campos no editables*
- *Rejillas de botones*
- *Botones 3D, botones opción 3D, casillas de selección 3D*
- *Menús desplegables, menús jerárquicos desplegables*
- *Menús/listas desplegables*
- *Menús imagen*
- *Áreas de desplazamiento*
- *Botones invisibles, botones inversos, botones opción imágenes*
- *Gráficos*
- *Objetos externos (para los cuales la entrada de datos es aceptada por el plug-in 4D)*
- *Pestañas*
- *Separadores.*

*Cuando los eventos On Getting Focus y/o On losing Focus son seleccionados para un objeto tabulable, usted puede detectar y administrar el cambio de foco, utilizando el comando **Form event** que devolverá On Getting Focus u On losing Focus, dependiendo del caso.*

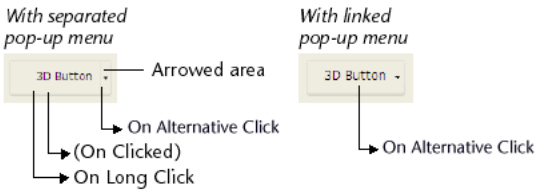
Botones 3D

Los botones 3D le permiten establecer interfaces gráficas avanzadas (para una descripción de los botones 3D, consulte el Manual de Diseño). Además de los eventos genéricos, dos eventos específicos pueden utilizarse para administrar estos botones:

- *On Long Click: este evento se genera cuando un botón 3D recibe un clic y el botón del ratón se mantiene presionado por un cierto periodo de tiempo. En teoría, el periodo de tiempo para el cual este evento se genera es igual a la longitud máxima de tiempo que separa un doble clic, como está definido en las preferencias del sistema.*
Este evento puede ser generado para todos los estilos de botones 3D, botones de opción 3D y casillas de selección 3D, con excepción de los botones 3D de "antigua generación" (estilo offset de fondo) y las áreas de flechas de botones 3D con menús desplegables (ver abajo).

Este evento se utiliza generalmente para mostrar menús desplegables en caso de clics largos en los botones. El evento On Clicked, si está seleccionado, se genera si el usuario libera el botón del ratón antes del tiempo límite de "clic largo".

- *On Alternative Click: algunos estilos de botones 3D pueden estar asociados a un menú desplegable y mostrar una flecha. Hacer clic en esta flecha hace que aparezca una caja de selección que ofrece un conjunto de acciones alternativas en relación con la acción principal del botón.*
4D le permite administrar este tipo de botón utilizando el evento On Alternative Click. Este evento se genera cuando el usuario hace clic en la "flecha" (tan pronto como el botón del ratón sea presionado):
- Si el menú desplegable es de tipo "separado," el evento sólo se genera cuando un clic ocurre en la parte del botón con la flecha.
- Si el menú desplegable es de tipo "enlazado," el evento se genera cuando ocurre un clic en cualquier parte del botón. Por favor tenga en cuenta que el evento On Long Click no puede generarse con este tipo de botón.



Los estilos de botones 3D, botones de opción 3D y casillas de selección 3D que aceptan la propiedad "Con menú desplegable" son: Ninguno, Botón de barra de herramientas, Bevel, Bevel redondeado y Office XP.

List box

Varios eventos formulario pueden utilizarse para administrar las diferentes características específicas de los list box:

- *On Before Data Entry: este evento se genera justo antes de que una celda de list box sea editada (antes de que el cursor de entrada aparezca). Este evento permite al desarrollador, por ejemplo, mostrar un texto diferente dependiendo de que el usuario esté en modo visualización o en modo edición.*
- *On Selection Change: este evento se genera cada vez que la selección actual de filas o de columnas del list box se modifica. Este evento también se genera para las listas de registros y listas jerárquicas.*
- *On Column Moved: este evento se genera cuando una columna del list box es movida por el usuario utilizando arrastrar y soltar. No se genera si la columna se arrastra y suelta en su ubicación inicial. El comando **LISTBOX MOVED COLUMN NUMBER** devuelve la nueva posición de la columna.*
- *On Row Moved: este evento se genera cuando una fila del list box es movida por el usuario utilizando arrastrar y soltar. No se genera si la fila es arrastrada y soltada en su ubicación inicial.*
- *On Column Resize: este evento se genera cuando el largo de una columna del list box es modificado por el usuario. A partir de 4D v16, el evento se activa "en vivo", es decir, se envía de forma continua durante el evento, durante el tiempo que el list box o la columna en cuestión estén siendo redimensionados. Este redimensionamiento se realiza manualmente por un usuario, o puede ocurrir como resultado del redimensionamiento del list box y su columna junto con la propia ventana del formulario (si el formulario se redimensiona manualmente o utilizando el comando **RESIZE FORM WINDOW**).*
  *Nota: el evento On Column Resize no se dispara cuando se cambia el tamaño de una columna "falsa" (para más información sobre columnas falsas, consulte **Tema Opciones de redimensionamiento**).*
- *On Expand y On Collapse: estos eventos se generan cuando una línea de list box jerárquico se despliega o contrae.*
- *On Header Click: este evento se genera cuando ocurre un clic en el encabezado de una columna en el list box. En este caso, el comando **Self** le permite conocer el encabezado de la columna en la que se hizo clic. El evento On Clicked se genera cuando un clic derecho (Windows) o Ctrl+clic (Mac OS) ocurre en una columna o en un encabezado de columna. Puede probar el número de clics realizados por el usuario utilizando el comando **Clickcount**.*
  *Si la propiedad **Ordenable** fue seleccionada para el list box, usted puede decidir si permite o no una ordenación estándar de la columna pasando el valor 0 ó -1 en la variable $0:*
  - Si $0 es igual a 0, se efectúa la ordenación estándar.
  - Si $0 es igual a -1, no se efectúa la ordenación estándar y el encabezado no muestra la flecha de ordenación. El desarrollador puede aún generar una ordenación de las columnas basada en criterios de ordenación personalizados utilizando los comandos de gestión de arrays de 4D.
  *Si la propiedad **Ordenable** no está seleccionada para el list box, la variable $0 no se utiliza.*
- *On Footer Click: este evento está disponible para el objeto list box o columna de list box. Se genera cuando ocurre un clic en el pie de un list box o de una columna de list box. En este caso, el comando **OBJECT Get pointer** devuelve un puntero a la variable del pie donde se hizo clic. El evento se genera para clic izquierdo y derecho.*
  *Puede probar el número de clics realizados por el usuario utilizando el comando **Clickcount**.*
- *On After Sort: este evento se genera justo después de que se efectúe una ordenación estándar (por ejemplo, no se genera si $0 devuelve -1 en el evento On Header Click). Este mecanismo es útil para conservar los sentidos de las últimas ordenaciones efectuadas por el usuario. En este evento, el comando **Self** devuelve un puntero a la variable del encabezado de la columna ordenada.*
- *On Delete Action: este evento se genera cada vez que el usuario intenta borrar los elementos seleccionados presionando la tecla suprimir (**Supr** o **Retroceder**) o seleccionando el comando **Eliminar** del menú **Edición**. Este evento sólo está disponible a nivel del objeto list box. Note que generar el evento es la única acción realizada por 4D: el programa no borra los elementos. El desarrollador debe manejar la eliminación y los mensajes de advertencia previos que se muestren.*
- *On Scroll (nuevo en v15): este evento se genera tan pronto como el usuario desplaza las filas o columnas de la lista. El evento sólo se genera cuando el desplazamiento es el resultado de una acción del usuario: utilizando barras y/o cursores de desplazamiento, la rueda del ratón o el teclado. No se genera cuando el desplazamiento es debido a la ejecución del comando **OBJECT SET SCROLL POSITION** .*
  Este evento se dispara después de cualquier otro evento de usuario relacionado con la acción de desplazamiento (On Clicked, On After Keystroke, etc.). El evento sólo se genera en el método objeto (no en el método formulario). Consulte el ejemplo 15.
- *On Alternative Click (nuevo en v15): este evento se genera en las columnas de list box de tipo array de objetos, cuando el usuario hace clic en un botón de puntos suspensivos de widget (atributo "alternateButton"). Para más información, consulte la sección **Utilizar arrays objetos en las columnas (4D View Pro)**.*

Dos eventos genéricos también pueden utilizarse en el contexto de un list box de tipo "selección":

- *On Open Detail: este evento se genera cuando un registro está apunto de ser mostrado en el formulario detallado asociado al list box de tipo "selección" (y antes de que este formulario se abra).*
- *On Close Detail: este evento se genera cuando un registro mostrado en el formulario detallado asociado al list box está a punto de cerrarse (sin importarle si el registro fue modificado o no).*

Listas jerárquicas

Además de los eventos genéricos, varios eventos específicos pueden utilizarse para administrar las acciones que los usuarios en las listas jerárquicas:

- *On Selection Change: este evento se genera cada vez que la selección en la lista jerárquica se modifica después de un clic o de que se presione una tecla.*
  Este evento también es generado en los objetos list box y listas de registros.

- *On Expand: este evento se genera cada vez que un elemento de la lista jerárquica se despliega con un clic o al presionar una tecla.*
- *On Collapse: este evento se genera cada vez que un elemento de la lista jerárquica se contrae con un clic o al presionar una tecla.*
- *On Delete Action: este evento se genera cada vez que el usuario intenta borrar los elementos seleccionados presionando una tecla de supresión (**Supr** o **Retroceso**) o seleccionando el comando **Eliminar** del menú **Edición**. Note que la generación del evento es la única acción efectuada por 4D: el programa no borra ningún elemento. El desarrollador debe encargarse de la eliminación y de todos los mensajes de alerta que aparezcan (ver el ejemplo).*

Estos eventos no son mutuamente exclusivos. Pueden generarse uno después del otro para una lista jerárquica:

- En respuesta a presionar una tecla (en orden):

Evento	**Contexto**
On Data Change	*Un elemento estaba en edición*
On Expand/*On Collapse*	*Apertura/cierre de una sublista utilizando las teclas de flechas -> o <-*
On Selection Change	*Selección de un nuevo elemento*
On Clicked	*Activación de la lista utilizando el teclado*

- En respuesta a un clic (en orden):

Evento	**Contexto**
On Data Change	*Un elemento estaba en edición*
On Expand/*On Collapse*	*Apertura/cierre de una sublista utilizando los iconos de despliegue/contracción o Doble-clic en una sublista no editable*
On Selection Change	*Selección de un nuevo elemento*
On Clicked / *On Double Clicked*	*Activación de la lista por un clic o un doble clic*

Variables y campos imagen

- *El evento formulario On Scroll se genera tan pronto como el usuario desplaza una imagen dentro del área (campo o variable) que lo contiene. Puede desplazar el contenido de una área imagen cuando el tamaño del área es menor a su contenido y el formato de visualización es **"Truncado (no centrado)"**. Para mayor información, consulte **Formatos imagen**.*
  *El evento se genera cuando el desplazamiento es el resultado de una acción del usuario: utilizando los cursores o las barras de desplazamiento, utilizando la rueda del ratón o las teclas de desplazamiento del teclado (para mayor información sobre el desplazamiento utilizando el teclado, consulte **Barras de desplazamiento**). Este evento no se genera cuando el objeto se desplaza por la ejecución del comando **OBJECT SET SCROLL POSITION** . Este evento se dispara luego de que un evento de usuario relacionado con la acción de desplazamiento (On Clicked, On After Keystroke, etc.). Se genera en el método objeto (no en el método formulario). Consulte el ejemplo 14.*

- *(Nuevo en v16) El evento On Mouse Up se genera cuando el usuario acaba de liberar el botón izquierdo del ratón mientras se arrastra en un área de imagen (campo o variable). Este evento es útil, por ejemplo, cuando se desea que el usuario pueda mover, redimensionar o dibujar objetos en un área SVG.*
  *Cuando se genera el evento On Mouse Up, puede obtener las coordenadas locales donde se soltó el botón del ratón. Estas coordenadas se devuelven en el **MouseX** y **MouseY** **Variables sistema**. Las coordenadas se expresan en píxeles con respecto a la esquina superior izquierda de la imagen (0,0).*
  *Al utilizar este evento, debe llamar al comando **Is waiting mouse up** para manejar los casos en que el gestor de estado del formulario podría estar desincronizado. Este es el caso, por ejemplo, cuando un cuadro de diálogo de alerta se muestra sobre el formulario, mientras que el botón del ratón no se ha liberado. Para más información y un ejemplo de uso del evento On Mouse Up, por favor consulte la descripción del comando **Is waiting mouse up**.*

  ***Nota:** si la opción "Arrastrable" se marca para el objeto imagen, el evento On Mouse Up nunca se genera.*

Subformularios

Un objeto contenedor de subformulario (objeto incluido en el formulario padre, contiene una instancia de subformulario) soporta los siguientes eventos:

- *On Load y On Unload: respectivamente apertura y cierre del subformulario (estos eventos también deben activarse a nivel de formulario padre para que sean tenidos en cuenta). Note que estos eventos se generan antes que los del formulario padre. Note también, que de acuerdo con los principios de funcionamiento de los eventos de formulario, si el subformulario se ubica en una página diferente a la página 0 ó 1, estos eventos sólo se generarán cuando la página se muestre/cierre (y no cuando el formulario se muestre/cierre).*
- *On Validate: validación de la entrada de datos en el subformulario.*
- *On Data Change: el valor de la variable del objeto contenedor del subformulario ha sido modificado.*
- *On Getting Focus y On Losing Focus: el contenedor del subformulario obtiene o pierde el foco. Estos eventos son generados en el método del objeto de subformulario cuando se seleccionan. Se envían al método de formulario del subformulario, lo que significa, por ejemplo, que puede administrar la visualización de los botones de navegación en el subformulario en función del foco.*
  Note que los objetos de subformulario pueden ellos mismos tener el foco.
- *On bound variable change: este evento específico se genera en el contexto del método de formulario del subformulario tan pronto como un valor se asigna a la variable asociada al subformulario en el formulario padre (incluso si el mismo valor es reasignado) y si el subformulario pertenece a la página del formulario actual o a la página 0. Para mayor información sobre la gestión de subformularios, consulte el manual de Diseño.*

***Nota:** es posible definir todo tipo de evento personalizado que puede ser generado en un subformulario vía el comando **CALL SUBFORM CONTAINER**. Este comando le permite llamar al método del objeto contenedor y pasarle un código de evento.*

***Nota:** los eventos On Clicked y On Double Clicked generados en el subformulario son recibidos en primer lugar por el método formulario del subformulario y luego por el método formulario del formulario local.*

Areas web

Hay siete eventos de formularios específicamente disponibles para las áreas web:

- *On Begin URL Loading: este evento se genera al inicio de la carga de un nuevo URL en el área Web. La variable "URL" asociada con el área web permite conocer el URL que se está cargando.*
  ***Nota:** el URL que se está cargando es diferente del URL actual (consulte la descripción del comando **WA Get current URL**).*
- *On URL Resource Loading: este evento se genera cada vez que se carga un nuevo recurso (imagen, marco, etc.) en la página web actual. La variable "Progression" asociada al área le permite buscar el estado actual de la carga.*
- *On End URL Loading: este evento se genera cuando se cargan todos los recursos del URL actual. Puede llamar el comando **WA Get current URL** para conocer el URL cargado.*
- *On URL Loading Error: este evento se genera cuando se detecta un error durante la carga de un URL. Puede llamar el comando **WA GET LAST URL ERROR** para obtener información sobre el error.*
- *On URL Filtering: este evento se genera cuando la carga de un URL es bloqueada por el área web debido a un filtro definido utilizando el comando **WA SET URL FILTERS**. Puede conocer el URL bloqueado utilizando el comando **WA Get last filtered URL**.*
- *On Open External Link: este evento se genera cuando la carga de un URL es bloqueada por el área Web y el URL se abre con el navegador del sistema actual, debido a un filtro definido utilizando el comando **WA SET EXTERNAL LINKS FILTERS**. Puede conocer el URL bloqueado utilizando el comando **WA Get last filtered URL**.*
- *On Window Opening Denied: este evento se genera cuando al abrir una ventana pop-up ha sido bloqueada para el área Web. Las áreas web 4D no permiten la apertura de ventanas pop-up. Puede conocer el URL bloqueado utilizando el comando **WA Get last filtered URL**.*

Áreas 4D View Pro

*El evento On VP Ready sólo está disponible para las áreas 4D View Pro. Se genera cuando se completa la carga del área 4D View Pro. Necesita utilizar este evento para escribir el código de inicialización del área. Para más información, consulte la sección **Evento formulario On VP Ready**.*

Ejemplo 1

*Este ejemplo muestra el evento **On Validate** utilizado para asignar automáticamente (a un campo) la fecha cuando el registro es modificado:*

```
  ` Método de un formulario
Case of
  ` ...
   :(Form event=On Validate)
      [unaTabla]Fecha de modificación:=Current date
End case
```

Ejemplo 2

En este ejemplo, la gestión completa de un menú desplegable, (inicialización, clics del usuario y liberación de objeto) está encapsulada en el método del objeto:

```
  ` Método de objeto del menú desplegable asTalla
Case of
   :(Form event=On Load)
      ARRAY TEXT(31;asTalla;3)
      asTalla{1}:="pequeña"
      asTalla{1}:="mediana"
      asTalla{1}:="grande"
   :(Form event=On Clicked)
      If(asTalla#0)
         ALERT("Escogió una hamburguesa "+asTalla{asTalla}")
      End if
   :(Form event=On Unload)
      CLEAR VARIABLE(asTalla)
End case
```

Ejemplo 3

Este ejemplo muestra cómo aceptar y administrar una operación de arrastrar y soltar para un objeto de campo que sólo acepta valores de imágenes, en un método de objeto,

```
  ` Método de objeto de campo Imagen [unaTabla]unaImagen
Case of
   :(Form event=On Drag Over)
  ` Ha comenzado una operación arrastrar y soltar y el ratón está sobre el campo
  ` Obtener la información sobre el objeto fuente
      DRAG AND DROP PROPERTIES($vpObjetoOrigen;$vlElementoOrigen;$lProcesoOrigen)
  ` Note que no necesitamos probar el número de proceso fuente
  ` para el método de objeto ejecutado ya que es el mismo proceso
      $vlTipoDatos:=Type($vpObjetoOrigen->)
```

```
   ` ¿Los datos fuente son una imagen (campo, variable o array)?
      If((($vlTipoDatos=Is picture)|($vlTipoDatos=Picture array))
   ` Si es así, acepte el arrastrar.
   ` Note que el botón del ratón aún está presionado, el único efecto mientras
   ` acepta el arrastrar es permitir a 4D resaltar el objeto de manera que el usuario
   ` sepa que los datos fuente pueden ser soltados en este objeto.
         $0:=0
      Else
   ` De lo contrario, rechace el arrastrar
         $0:=-1
   ` En este caso, el objeto no se resalta
      End if
   :(Form event=On Drop)
   ` Los datos fuente han sido soltados sobre el objeto, por lo tanto necesitamos copiarlos en el objeto
   ` Obtener la información sobre el objeto fuente
      DRAG AND DROP PROPERTIES($vpObjetoOrigen;$vlElementoOrigen;$lProcesoOrigen)
      $vlTipoDatos:=Type($vpObjetoOrigen->)
      Case of
   ` El objeto fuente es un campo o una variable tipo imagen
         :($vlTipoDatos=Is picture)
   ` El objeto fuente es del mismo proceso (de la misma ventana y formulario)?
            If($lProcesoOrigen=Current process)
   ` Si es así, copiar el valor fuente
               [aTabla]almagen:=$vpObjetoOrigen->
            Else
   ` Si no, ¿es el objeto fuente una variable?
               If(Is a variable($vpObjetoOrigen))
   ` Si es así, obtener el valor del proceso fuente
                  GET PROCESS VARIABLE($lProcesoOrigen;$vpObjetoOrigen->;$vgImagenArrastrada)
                  [aTabla]almagen:=$vgImagenArrastrada
               Else
   ` Si no, utilice CALL PROCESS para obtener el valor del campo del proceso fuente
               End if
            End if
   ` El objeto fuente es un array de imágenes
         :($vlTipoDatos=Picture array)
   ` ¿Está el objeto fuente en el mismo proceso (en la misma ventana y formulario)?
            If($lProcesoOrigen=Current process)
   ` Si es así, copiar el valor fuente
               [aTabla]almagen:=$vpObjetoOrigen->{$vlElementoOrigen}
            Else
   ` Sino, obtener el valor del proceso fuente
               GET PROCESS VARIABLE($lProcesoOrigen;$vpObjetoOrigen;
               ->{$vlElementoOrigen};$vgImagenArrastrada)
               [aTabla]almagen:=$vgImagenArrastrada
            End if
      End case
End case
```

Nota: *para más ejemplos sobre gestión de los eventos* ***On Drag Over*** *y* ***On Drop****, consulte los ejemplos del comando* ***DRAG AND DROP PROPERTIES****.*

Ejemplo 4

Este ejemplo es una plantilla para un método de formulario. Muestra cada uno de los posibles eventos que pueden ocurrir cuando un informe utiliza un formulario como formulario de salida:

```
   ` Método de un formulario utilizado como formulario de salida de un informe
$vpFormTabla:=Current form table
Case of
   ` ...
   :(Form event=On Header)
   ` Un área de encabezado está apunto de imprimirse
      Case of
         :(Before selection($vpFormTabla->))
   ` El código para la primera ruptura de encabezado debe estar aquí
         :(Level=1)
   ` El código para la ruptura de encabezado de nivel 1 debe estar aquí
         :(Level=2)
   ` El código para la ruptura de encabezado de nivel 2 debe estar aquí
   ` ...
```

```
      End case
   :(Form event=On Printing Detail)
   ` Se va a imprimir un registro
   ` El código para cada registro debe ir aquí
   :(Form event=On Printing Break)
   ` Un área de ruptura está a punto de imprimirse
      Case of
         :(Level=0)
   ` El código para la ruptura 0 debe estar aquí
         :(Level=1)
   ` El código para la ruptura 1 debe estar aquí
   ` ...
      End case
   :(Form event=On Printing Footer)
      If(End selection($vpFormTabla->))
   ` El código para el último pie de página va aquí
      Else
   ` El código para el pie de página debe ir aquí
      End if
End case
```

Ejemplo 5

*Este ejemplo muestra la plantilla de un método de formulario que administra los eventos que pueden ocurrir en un formulario mostrado utilizando los comandos **DISPLAY SELECTION** o **MODIFY SELECTION**. Por propósitos didácticos, muestra la naturaleza del evento en la barra de título de la ventana del formulario.*

```
   ` Un método de formulario
Case of
   :(Form event=On Load)
      $vsEvento:="El formulario va a ser visualizado"
   :(Form event=On Unload)
      $vsEvento:="El formulario de salida ha sido cerrado y va a desaparecer de la pantalla"
   :(Form event=On Display Detail)
      $vsEvento:="Mostrando el registro #"+String(Selected record number([LaTabla]))
   :(Form event=On Menu Selected)
      $vsEvento:="Un comando de menú ha sido seleccionado"
   :(Form event=On Header")
      $vsEvento:="El área de encabezado está a punto de ser dibujada"
   :(Form event=On Clicked")
      $vsEvento:="Se ha hecho clic en un registro"
   :(Form event=On Double Clicked")
      $vsEvento:="Se ha hecho doble clic en un registro"
   :(Form event=On Open Detail)
      $vsEvento:="Se hizo doble clic en el registro #"+String(Selected record number([LaTabla]))
   :(Form event=On Close Detail)
      $vsEvento:="Regreso al formulario de salida"
   :(Form event=On Activate)
      $vsEvento:="La ventana de formulario pasa al primer plano"
   :(Form event=On Deactivate)
      $vsEvento:="La ventana del formulario no es más la ventana del primer plano"
   :(Form event=On Menu Selected)
      $vsEvento:="Se ha seleccionado un elemento del menú"
   :(Form event=On Outside Call)
      $vsEvento:="Se ha recibido una llamada de exterior"
   Else
      $vsEvento:="¿Qué pasa? El evento #"+String(Form event)
End case
SET WINDOW TITLE($vsEvento)
```

Ejemplo 6

*Para los ejemplos sobre gestión de los eventos On Before Keystroke y On After Keystroke, ver los ejemplos de los comandos **Get edited text**, **Keystroke** y **FILTER KEYSTROKE**.*

Ejemplo 7

Este ejemplo muestra cómo tratar de la misma forma los clics y doble clic en un área de desplazamiento:

```
   ` Método de objeto para el área de desplazamiento asOpciones
Case of
   :(Form event=On Load)
      ARRAY STRING(...;asOpciones;...)
` ...
      asOpciones:=0
   :((Form event=On Clicked)|(Form event=On Double Clicked))
      If(asOpciones#0)
` Se ha hecho clic en un elemento, hacer algo aquí
` ...
      End if
` ...
End case
```

Ejemplo 8

Este ejemplo muestra cómo tratar los clics y doble clics utilizando una respuesta diferente. Note el uso del elemento cero para conservar el valor del elemento seleccionado:

```
   ` Método de objeto para el área de desplazamiento asChoices
Case of
   :(Form event=On Load)
      ARRAY TEXT(...;asOpciones;...)
` ...
      asOpciones:=0
      asOpciones{0}:="0"
   :(Form event=On Clicked)
      If(asOpciones#0)
         If(asOpciones#Num(asOpciones))
` Se ha hecho clic en un nuevo elemento, hacer algo aquí
` ...
` Guardar el nuevo elemento seleccionado para la próxima vez
               asOpciones{0}:=String(asOpciones)
         End if
      Else
         asOpciones:=Num(asOpciones{0})
      End if
   :(Form event=On Double Clicked)
      If(asOpciones#0)
` Se ha hecho doble clic sobre un elemento, hacer algo diferente aquí
      End if
` ...
End case
```

Ejemplo 9

Este ejemplo muestra cómo mantener un área de texto a partir de un método desde un método de formulario, utilizando los eventos On Getting Focus y On losing Focus:

```
  `Método de formulario [Contactos];"Entrada"
Case of
   :(Form event=On Load)
      C_TEXT(vtAreaEstado)
      vtAreaEstado:=""
   :(Form event=On Getting Focus)
      RESOLVE POINTER(Focus object;$vsNombreVar;$vlNumTabla;$vlNumCampo)
      If(($vlNumTabla#0) & ($vlNumCampo#0))
         Case of
            :($vlNumCampo=1) ` Campo nombre
               vtAreaEstado:="Introduzca el nombre del contacto; se pasará automáticamente a mayúsculas"
` ...
            :($vlNumCampo=10) ` Campo código postal
               vtAreaEstado:="Introduzca un código postal; será verificado y validado automáticamente"
` ...
         End case
      End if
   :(Form event=On Losing Focus)
      vtAreaEstado:=""
```

```
   ` ...
End case
```

Ejemplo 10

Este ejemplo muestra cómo responder al evento de cierre de una ventana con un formulario utilizado para la entrada de datos:

```
   ` Método para un formulario de entrada
$vpFormTabla:=Current form table
Case of
   ` ...
   :(Form event=On Close Box)
      If(Modified record($vpFormTabla->))
         CONFIRM("Este registro ha sido modificado. ¿Quiere guardar los cambios?")
         If(OK=1)
            ACCEPT
         Else
            CANCEL
         End if
      Else
         CANCEL
      End if
   ` ...
End case
```

Ejemplo 11

Este ejemplo muestra cómo pasar a mayúsculas un campo de tipo texto o alfanumérico cada vez que el valor se modifique:

```
   `Método de objeto para [Contactos]Nombre
Case of
   ` ...
   :(Form event=On Data Change)
      [Contactos]Nombre:=Uppercase(Substring([Contactos]Nombre;1;1))+
      Lowercase(Substring([Contactos]Nombre;2))
   ` ...
End case
```

Ejemplo 12

Este ejemplo muestra cómo pasar a mayúsculas un campo de tipo texto o alfanumérico cada vez que el valor se modifique:

```
   `Método de objeto para [Contactos]Nombre
Case of
   ` ...
   :(Form event=On Data Change)
      [Contactos]Nombre:=Uppercase(Substring([Contactos]Nombre;1;1))+
      Lowercase(Substring([Contactos]Nombre;2))
   ` ...
End case
```

Ejemplo 13

El siguiente ejemplo ilustra cómo manejar una acción de eliminación en una lista jerárquica:

```
   ... //método de la lista jerárquica
:($Event=On Delete Action)
 ARRAY LONGINT($itemsArray;0)
 $Ref:=Selected list items(<>HL;$itemsArray;*)
 $n:=Size of array($itemsArray)

 Case of
    :($n=0)
       ALERT("Ningún elemento seleccionado")
       OK:=0
    :($n=1)
       CONFIRM("¿Quiere eliminar este elemento?")
```

```
    :($n>1)
        CONFIRM("¿Quiere eliminar estos elementos?")
End case

If(OK=1)
    For($i;1;$n)
        DELETE FROM LIST(<>HL;$itemsArray{$i};*)
    End for
End if
```

Ejemplo 14

En este ejemplo, el evento formulario On Scroll permite sincronizar la visualización de dos imágenes en un formulario. El siguiente código se añade en el método del objeto "satélite" (campo imagen o variable image):

```
Case of
    :(Form event=On Scroll)
 // tomamos la posición de la imagen de la izquierda
        OBJECT GET SCROLL POSITION(*;"satellite";vPos;hPos)
 // y la aplicamos a la imagen de la derecha
        OBJECT SET SCROLL POSITION(*;"plan";vPos;hPos;*)
End case
```

Resultado:

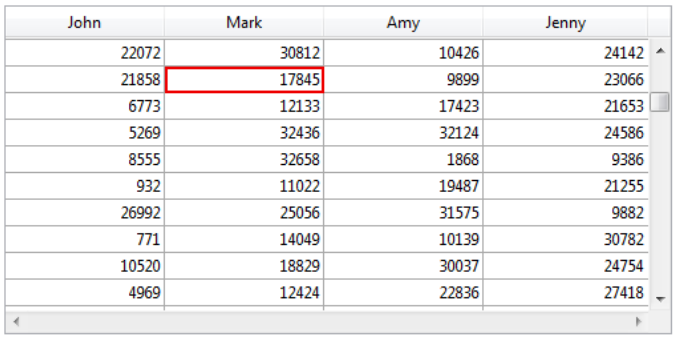
Ejemplo 15

Usted desea dibujar un rectángulo rojo alrededor de la celda seleccionada de un list box y desea que el rectángulo se mueva junto con el list box si es desplazado verticalmente por el usuario. En el método objeto del list box, puede escribir:

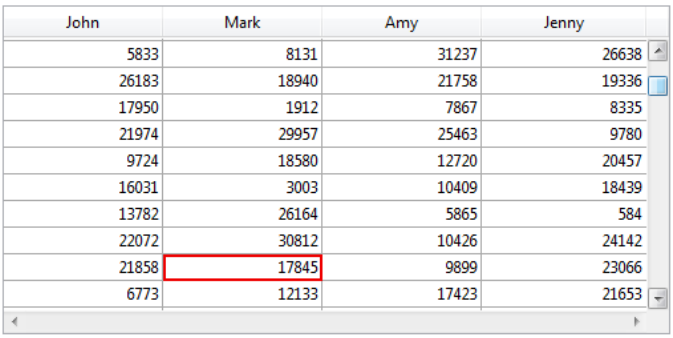
```
Case of

    :(Form event=On Clicked)
        LISTBOX GET CELL POSITION(*;"LB1";$col;$raw)
        LISTBOX GET CELL COORDINATES(*;"LB1";$col;$raw;$x1;$y1;$x2;$y2)
        OBJECT SET VISIBLE(*;"RedRect";True)  //inicializa un rectángulo rojo
        OBJECT SET COORDINATES(*;"RedRect";$x1;$y1;$x2;$y2)

    :(Form event=On Scroll)
        LISTBOX GET CELL POSITION(*;"LB1";$col;$raw)
        LISTBOX GET CELL COORDINATES(*;"LB1";$col;$raw;$x1;$y1;$x2;$y2)
        OBJECT GET COORDINATES(*;"LB1";$xlb1;$ylb1;$xlb2;$ylb2)
        $toAdd:=LISTBOX Get headers height(*;"LB1") //tener en cuenta la altura del encabezado para que no lo sobrepase
        If($ylb1+$toAdd<$y1) & ($ylb2>$y2) //si estamos dentro del list box
  //para simplificar, sólo manejamos encabezados
  //pero debemos manejar clipping horizontal
  //así como también las barras de desplazamiento
            OBJECT SET VISIBLE(*;"RedRect";True)
            OBJECT SET COORDINATES(*;"RedRect";$x1;$y1;$x2;$y2)
        Else
            OBJECT SET VISIBLE(*;"RedRect";False)
        End if
```

```
End case
```

Como resultado, el rectángulo rojo sigue el desplazamiento del list box:



John	Mark	Amy	Jenny
22072	30812	10426	24142
21858	17845	9899	23066
6773	12133	17423	21653
5269	32436	32124	24586
8555	32658	1868	9386
932	11022	19487	21255
26992	25056	31575	9882
771	14049	10139	30782
10520	18829	30037	24754
4969	12424	22836	27418



John	Mark	Amy	Jenny
5833	8131	31237	26638
26183	18940	21758	19336
17950	1912	7867	8335
21974	29957	25463	9780
9724	18580	12720	20457
16031	3003	10409	18439
13782	26164	5865	584
22072	30812	10426	24142
21858	17845	9899	23066
6773	12133	17423	21653

In break

In break -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*Devuelve True si el ciclo de ejecución es in break*

Descripción

In break *devuelve True para el ciclo de ejecución In break.*

Para que se genere el ciclo de ejecución ***In break*** *asegúrese de que la propiedad del evento On Printing Break para el formulario y/o los objetos haya sido seleccionada en el entorno Diseño.*

Nota: *este comando es equivalente a utilizar* ***Evento formulario*** *y probar si devuelve el evento On Printing Break.*

In footer

In footer -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*Devuelve True si el ciclo de ejecución es in footer*

Descripción

In footer *devuelve True para el ciclo de ejecución In footer.*

Para que se genere el ciclo de ejecución ***In footer*** *asegúrese de que la propiedad del evento On Printing footer para el formulario y/o los objetos haya sido seleccionada en el entorno Diseño.*

Nota: *este comando es equivalente a utilizar* ***Evento formulario*** *y probar si devuelve el evento On Printing footer.*

In header

In header -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*Devuelve True si el ciclo de ejecución es in header*

Descripción

In header *devuelve True para el ciclo de ejecución In header.*

Para que se genere el ciclo de ejecución ***In header****, asegúrese de que la propiedad del evento On Header para el formulario y/o los objetos haya sido seleccionada en el entorno Diseño.*

Nota: *este comando es equivalente a utilizar* ***Evento formulario*** *y probar si devuelve el evento On Header.*

Is waiting mouse up

Is waiting mouse up -> Resultado

Parámetro	Tipo	Descripción
Resultado	Booleano	True si el objeto está esperando por un evento mouse up, de lo contrario False

Descripción

Tema: Eventos formulario

El comando **Is waiting mouse up** devuelve **True** después de que el objeto actual haya hecho clic y el botón del ratón no se ha liberado, y cuando el diálogo aún tiene el foco. Debe llamarse desde el método de objeto del objeto actual.

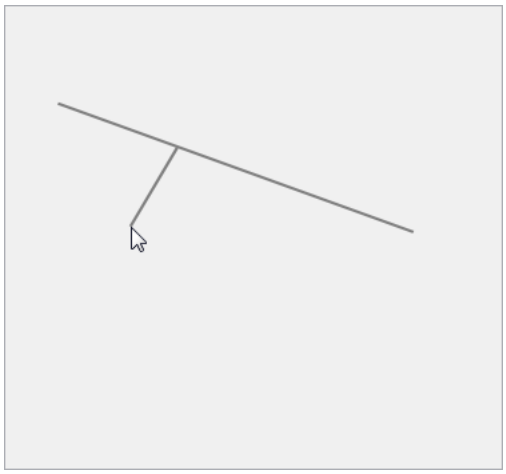
El comando devuelve **False** en los siguientes casos:

- no se llama a partir de un método de objeto
- o no se disparó el evento On Clicked en el objeto
- o la caja de diálogo ha perdido el foco.

Este comando debe ser utilizado junto con . Le permite sincronizar el estado interno del objeto de formulario con la aplicación global. Básicamente, permite a su código manejar el caso donde el usuario hizo clic y empezó a mover algo dentro de una imagen objeto de formulario, y esta acción es interrumpida por un evento externo, como un cuadro de diálogo de alerta. En este caso, el estado interno del objeto puede ser suspendido de forma indefinida debido a que se espera un evento mouse up que no ocurrirá. Para abordar este problema, debe proteger el código de movimiento del ratón con un comando **Is waiting mouse up** lo que le asegura que es ejecutado en un contexto válido.

Ejemplo

Usted desea que el usuario dibuje líneas en un área SVG. El nombre de la variable de área es vPictSvg:



Los eventos **On Mouse Up, On Mouse Move** y **On Clicked** han sido seleccionados para el objeto. El método de objeto contiene el siguiente código:

```
C_LONGINT($val1;$val2)
C_LONGINT(vLtracking) //bandera para el modo tracking
Case of
   :(Form event=On Clicked)
      If(Not(Contextual click) & Is waiting mouse up) //clic izquierdo únicamente
         If((MouseX#-1) & (MouseY#-1)) //estamos en el objeto
            vLtracking:=1 //estamos en modo tracking
            LineRef:=SVG_New_line(svgRef;MouseX;MouseY;MouseX;MouseY;"gray";3) //crear una línea fantasma
            SVG_SET_ID(LineRef;"ghostLine") //dar un id a la línea fantasma
            vPictSvg:=SVG_Export_to_picture(svgRef)
         End if
      End if
   :(Form event=On Mouse Move)
      If(vLtracking=1)
         If(Not(Is waiting mouse up)) //un event canceló el dibujo
            SVG_CLEAR(LineRef) //eliminar la línea fantasma
            vPictSvg:=SVG_Export_to_picture(svgRef) //guardar la línea real
            vLtracking:=0 //detener el modo tracking
         Else //el objeto está esperando por un mouse up
            If((MouseX#-1) & (MouseY#-1))
               SVG SET ATTRIBUTE(vPictSvg;"ghostLine";"x2";MouseX;"y2";MouseY;*)
            End if
```

```
            End if
        End if
    :(Form event=On Mouse Up)
        If(MouseX=-1)
            SVG GET ATTRIBUTE(vPictSvg;"ghostLine";"x2";MouseX)
        End if
        If(MouseY=-1)
            SVG GET ATTRIBUTE(vPictSvg;"ghostLine";"y2";MouseY)
        End if
        SVG GET ATTRIBUTE(vPictSvg;"ghostLine";"x1";$val1 ) //obtener las coordenadas fanstamas
        SVG GET ATTRIBUTE(vPictSvg;"ghostLine";"y1";$val2 )
        SVG_CLEAR(LineRef) //delete the ghost line
        SVG_New_line(svgRef;$val1;$val2;MouseX;MouseY;"gray";3) //crear la línea actual
        vPictSvg:=SVG_Export_to_picture(svgRef) //guardar la línea actual
        vLtracking:=0
End case
```

Outside call

Outside call -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		Devuelve True si el ciclo de ejecución es outside call

Descripción

Outside call devuelve True para el ciclo de ejecución After.

Para que se genere el ciclo de ejecución **Outside call**, asegúrese de que la propiedad del evento On Outside call del formulario y/o los objetos se haya seleccionado en el entorno Diseño.

Nota: este comando es equivalente a utilizar **Evento formulario** y probar si devuelve el evento On Outside call.

POST OUTSIDE CALL

POST OUTSIDE CALL (proceso)

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número de proceso

Compatibility Note

*This command was named **CALL PROCESS** in previous 4D releases.*

Descripción

***POST OUTSIDE CALL** llama al formulario mostrado en la ventana del primer plano de proceso.*

***Importante: POST OUTSIDE CALL** sólo funciona entre procesos que se ejecutan en el mismo equipo.*

Si llama a un proceso que no existe, no pasa nada.

*Si proceso (el proceso llamado) no está mostrando un formulario actualmente, no pasa nada. El formulario mostrado en el proceso llamado recibe un evento On Outside Call. Este evento debe haber sido seleccionado para ese formulario en la ventana **Propiedades del formulario** del entorno Diseño, y usted debe administrar el evento en el método de formulario. Si el evento no está seleccionado o si no es administrado en el método de formulario, el comando no hace nada.*

***Nota:** la recepción del evento On Outside Call en un formulario de entrada provoca el cambio del contexto de entrada del formulario. En particular, si un campo estaba siendo editado, se genera el evento On Data Change.*

*El proceso llamante (el proceso en el cual el comando **POST OUTSIDE CALL** se ejecuta) no "espera", **POST OUTSIDE CALL** tiene un efecto inmediato. Si es necesario, debe escribir un bucle de espera para tratar una eventual respuesta del proceso llamante, utilizando las variables interproceso o las variables proceso (reservadas para este propósito) que pueden ser leídas y escritas entre los dos procesos (utilizando **GET PROCESS VARIABLE** y **SET PROCESS VARIABLE**).*

*Para comunicarse entre procesos que no muestran formularios, utilice los comandos **GET PROCESS VARIABLE** y **SET PROCESS VARIABLE**.*

***Tip: POST OUTSIDE CALL** acepta la sintaxis alterna **POST OUTSIDE CALL**(-1). Para no volver lenta la ejecución de los métodos, 4D no rediseña las variables interproceso cada vez que son modificadas. Si pasa -1 en lugar de un número de referencia de proceso en el parámetro proceso, 4D no llama ningún proceso. En lugar de eso, rediseña todas las variables interproceso mostradas actualmente en todas las ventanas de todos los procesos que se ejecutan en el mismo equipo.*

Ejemplo

Ver el ejemplo de On Exit Database Method.

Right click

Right click -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*True si se detecta un clic derecho, si no False*

Descripción

*El comando **Right click** devuelve **True** si se hace clic en el botón derecho del ratón.*

*Este comando debe utilizarse únicamente en el contexto del evento de formulario **On clicked**. Por lo tanto es necesario verificar en modo Diseño que el evento haya sido seleccionado correctamente en las propiedades del formulario y/o del objeto específico.*

SET TIMER

SET TIMER (ticCont)

Parámetro	Tipo		Descripción
ticCont	Entero largo	➡	Número de tics

Descripción

*El comando **SET TIMER** permite activar el evento de formulario On Timer y fijar, para el proceso y formulario actual, el número de tics (1 tic = 1/60 de segundo) entre cada evento de formulario On Timer.*

***Nota:** para mayor información sobre este evento de formulario, consulte la descripción del comando **Form event**.*

Este comando no tendrá efecto si se llama en un contexto en el que no muestra un formulario.

***Nota:** cuando el comando **SET TIMER** se ejecuta en el contexto de un subformulario (método de formulario del subformulario), se genera el evento On Timer en el subformulario y no al nivel del formulario padre.*

Si pasa -1 en el parámetro tickCount, el comando activará el evento de formulario On Timer "tan pronto como sea posible", en otras palabras, tan pronto como la aplicación 4D tome el control del administrador de eventos. Este principio permite asegurar que un formulario se muestre completamente antes de iniciar un proceso (fluidez de la aplicación).

*Para desactivar por programación el disparador del evento de formulario On Timer, llame nuevamente a **SET TIMER** y pase 0 en ticCont.*

Ejemplo

Imaginemos que usted quiere, cuando un formulario aparece en pantalla, que el ordenador haga bip cada tres segundos. Para hacer esto, escriba el siguiente método de formulario:

```
If(Form event=On Load)
   SET TIMER(60*3)
End if

If(Form event=On Timer)
   BEEP
End if
```

_o_During

_o_During -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		Devuelve True para el ciclo de ejecución In break

Nota de compatibilidad

No se recomienda el uso de esta antigua función de ciclo de ejecución genérico. Se recomienda utilizar ***Evento formulario*** *y probar los eventos específicos devueltos, como On Clicked.*

Fechas y horas

- *Add to date*
- *Current date*
- *Current time*
- *Date*
- *Day number*
- *Day of*
- *Milliseconds*
- *Month of*
- *SET DEFAULT CENTURY*
- *Tickcount*
- *Time*
- *Time string*
- *Timestamp*
- *Year of*

Add to date

Add to date (fecha ; años ; meses ; días) -> Resultado

Parámetro	Tipo		Descripción
fecha	Fecha	➡	Fecha a la cual añadir días, meses y años
años	Entero largo	➡	Número de años a añadir a la fecha
meses	Entero largo	➡	Número de meses a añadir a la fecha
días	Entero largo	➡	Número de días a añadir a la fecha
Resultado	Fecha	↩	Fecha resultante

Descripción

*El comando **Add to date** añade años, meses, y días a la fecha pasada en fecha, luego devuelve la fecha resultante.*
*Aunque usted puede utilizar los **Operadores de fechas** para añadir días a una fecha, **Add to date** le permite rápidamente añadir meses y años sin tener que lidiar con el número de días al mes o años bisiestos (como lo haría cuando utiliza el operador + en fechas).*

Ejemplo

```
` Esta línea calcula la fecha dentro de un año, el mismo día
$vdInUnaño:=Add to date(Current date;1;0;0)

` Esta línea calcula la fecha el próximo mes, el mismo día
$vdProximoMes:=Add to date(Current date;0;1;0)

` Esta línea hace lo mismo que $vdMañana:=Current date+1
$vdMañana:=Add to date(Current date;0;0;1)
```

Current date

Current date {(*)} -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Devuelve la fecha actual del servidor
Resultado	Fecha	➲	Fecha actual

Descripción

*El comando **Current date** devuelve la fecha actual tal como está definida en el reloj del sistema.*
***4D Server:** si pasa el parámetro asterisco (*) durante la ejecución de esta función en un equipo cliente 4D Client, la función devuelve la fecha actual del servidor.*

Ejemplo 1

El siguiente ejemplo muestra una caja de diálogo de alerta con la fecha actual:

```
ALERT("The date is "+String(Current date)+".")
```

Ejemplo 2

Si desarrolla una aplicación para el mercado internacional, usted necesita saber si la versión de 4D con la cual se ejecuta su aplicación funciona con las fechas con formato MM/DD/YYYY (versión US) o DD/MM/YYYY (versión francesa). Esta información es útil para la personalizar correctamente las áreas de entrada.
El siguiente método de proyecto permite hacerlo:

```
  ` Función global Sys date format
  ` Sys date format -> Cadena
  ` Sys date format -> Formato de datos 4D por defecto

C_STRING(31;$0;$vsDate;$vsMDY;$vsMonth;$vsDay;$vsYear)
C_LONGINT($1;$vlPos)
C_DATE($vdDate)

 `Obtener una fecha en la cual los valores de mes, día y año sean todos diferentes
$vdDate:=Current date
Repeat
   $vsMonth:=String(Month of($vdDate))
   $vsDay:=String(Day of($vdDate))
   $vsYear:=String(Year of($vdDate)%100)
   If(($vsMonth=$vsDay)|($vsMonth=$vsYear)|($vsDay=$vsYear))
      vOK:=0
      $vdDate:=$vdDate+1
   Else
      vOK:=1
   End if
Until(vOK=1)
$0:="" ` Inicialización del resultado de la función
$vsDate:=String($vdDate)
$vlPos:=Position("/";$vsDate) ` Buscar el primer separador / en la cadena ..
$vsMDY:=Substring($vsDate;1;$vlPos-1) ` Extraer los primeros dígitos de la fecha
$vsDate:=Substring($vsDate;$vlPos+1) ` Eliminar los primeros digitos y el primer separador /
Case of
   :($vsMDY=$vsMonth) ` Los dígitos expresan el mes
      $0:="MM"
   :($vsMDY=$vsDay) `Los dígitos expresan el día
      $0:="DD"
   :($vsMDY=$vsYear) `Los dígitos expresan el año
      $0:="YYYY"
End case
$0:=$0+"/" `Iniciar la construcción del resultado de la función
$vlPos:=Position("/";$vsDate) ` Buscar el segundo separador en la cadena ../..
$vsMDY:=Substring($vsDate;1;$vlPos-1) ` Extraer los siguientes dígitos de la fecha
$vsDate:=Substring($vsDate;$vlPos+1) ` Reducir la cadena a los últimos dígitos de la fecha
Case of
   :($vsMDY=$vsMonth) ` Los dígitos expresan el mes
```

```
        $0:=$0+"MM"
    :($vsMDY=$vsDay) ` Los dígitos expresan el día
        $0:=$0+"DD"
    :($vsMDY=$vsYear) ` Los dígitos expresan el año
        $0:=$0+"YYYY"
End case
$0:=$0+"/" `Iniciar la construcción del resultado de la función
Case of
    :($vsDate=$vsMonth) ` Los dígitos expresan el mes
        $0:=$0+"MM"
    :($vsDate=$vsDay) ` Los dígitos expresan el día
        $0:=$0+"DD"
    :($vsDate=$vsYear) ` Los dígitos expresan el año
        $0:=$0+"YYYY"
End case
 `En este momento $0 es igual a MM/DD/YYYY o DD/MM/YYYY o...
```

Current time

Current time {(*)} -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Devuelve la hora actual del servidor
Resultado	Hora	➲	Hora actual

Descripción

*El comando **Current time** devuelve la hora actual del reloj de sistema.*

*La hora actual siempre está entre 00:00:00 y 23:59:59. Utilice **String** o **Time string** para convertir en cadena la expresión de tipo hora devuelta por **Current time**.*

***4D Server:** si utiliza el parámetro (*) cuando ejecuta esta función en un equipo 4D Client, la función devuelve la hora actual del servidor.*

Ejemplo 1

*El siguiente ejemplo le muestra cómo medir la duración de una operación. Acá, **OperacionLarga** es un método que necesita ser cronometrado:*

```
$vhHoraInicio:=Current time  ` Guardar la hora de inicio
OperacionLarga  ` Realizar la operación
ALERT("La operación tomó "+String(Current time-$vhHoraInicio))  ` Mostrar la duración de la operación
```

Ejemplo 2

El siguiente ejemplo extrae las horas, minutos y segundos de la hora actual:

```
$vhAhora:=Current time
ALERT("La hora actual es: "+String($vhAhora\3600))
ALERT("El minuto actual es: "+String(($vhAhora\60)%60))
ALERT("El segundo actual es: "+String($vhAhora%60))
```

Date

Date (fechaCadena) -> Resultado

Parámetro	Tipo		Descripción
fechaCadena	Cadena	➡	Cadena que contiene la fecha a devolver
Resultado	Fecha	⮌	Fecha

Descripción

*El comando **Date** evalúa fechaCadena y devuelve una fecha.*

El parámetro fechaCadena debe respetar el formato fecha ISO o los parámetros regionales del sistema.

Formato fecha ISO

*La cadena debe estar en el formato: "AAAA-MM-DD**T**HH:MM:SS", por ejemplo "2013-11-20T10:20:00". En este caso, **Date** evalúa el parámetro fechaCadena correctamente, sin importar la configuración de lenguaje actual. Los decimales de segundos, precedidos por un punto, se soportan (ejemplo: "2013-11-20T10:20:00.9854").*

Si el formato fechaCadena no respeta este esquema ISO, luego la fecha se evalúa como un formato fecha corto basado en los parámetros regionales del sistema.

***Nota:** a partir de 4D v14, se recomienda utilizar el formato "YYYY-MM-DDTHH:MM:SS**Z**", conforme a la norma ISO y permitiéndole expresar la zona horaria.*

Parámetros regionales

*Si fechaCadena no corresponde al formato ISO, los parámetros regionales definidos en el sistema operativo para una fecha corta son utilizados para la evaluación. Por ejemplo, en la versión en español de 4D, por defecto la fecha debe estar en el orden MM/DD/AA (mes, día, año). El mes y el día pueden tener uno o dos dígitos. El año puede ser de dos o cuatro dígitos. Si el año es de dos dígitos, entonces **Date** considera si la fecha pertenece al siglo 20 o 21 en función del valor introducido. Por defecto el valor pivote es 30:*

- *si el valor introducido es superior o igual a 30, 4D considera que la fecha pertenece al siglo 20 y añade 19 delante del valor.*
- *si el valor introducido es inferior a 30, 4D considera que la fecha pertenece al siglo 21 y añade 20 delante del valor.*

*Este mecanismo puede configurarse utilizando el comando **SET DEFAULT CENTURY**.*
Los siguientes caracteres son separadores de fecha válidos: barra oblicua (/), espacio, punto (.), coma (,) y guión (-).

- *Si se pasa una fecha invalida (tal como "13/35/94" o "aa/12/94") en fechaCadena, **Date** devolverá una fecha vacía (!00/00/00!). Es su responsabilidad verificar que fechaCadena sea una fecha válida.*
- *Si la expresión fechaCadena se evalúa como indefinida, **Date** devuelve una fecha vacía (!00/00/00!). Esto es útil cuando se espera que el resultado de una expresión (por ejemplo, un atributo objeto) sea una fecha, incluso si puede ser indefinido.*

***Nota**: a partir de 4D v16 R6, las fechas pueden almacenarse en atributos objeto como valores de tipo de fecha. En versiones anteriores, solo podían almacenarse como cadenas (para más información sobre esta opción, consulte la sección [#title id="3239"/], "Utilizar el tipo fecha en lugar del formato fecha ISO en los objetos"). Para saber si un atributo contiene una fecha almacenada como una cadena o como una fecha, debe usar el comando **Value type** (ver el último ejemplo).*

Ejemplo 1

El siguiente ejemplo utiliza una caja para que el usuario introduzca una fecha. La cadena introducida por el usuario se convierte en una fecha y se guarda en la variable reqFecha:

```
vdReqFecha:=Date(Request("Por favor introduzca una fecha:";String(Current date)))
If(OK=1)
   ` Hacer algo con la fecha guardada en vdReqFecha
End if
```

Ejemplo 2

Los siguientes ejemplos muestran varios casos:

```
vdDate:=Date("12/25/94") //12/25/94 on a US system
vdDate2:=Date("40/40/94") //00/00/00
vdDate3:=Date("It was the 6/30/2016") //06/30/16
C_OBJECT($vobj)
$vobj:=New object("expDate";"2020-11-17T00:00:00.0000")
vdDate4:=Date($vobj.expDate) //11/17/20
vdDate5:=Date($vobj.creationDate) //00/00/00
```

Ejemplo 3

Fecha de evaluación basada en una fecha en formato ISO:

```
$vtDateISO:="2013-06-05T20:00:00"
$vDate:=Date($vtDateISO)
 //$vDate representa el 5 de junio de 2013 sin importar el lenguaje del sistema
```

Ejemplo 4

Usted desea obtener una fecha de un atributo objeto, sea cual sea la opción de almacenamiento de fecha del atributo actual:

```
If(Value type($myObj.myDate)=ls date)  //se almacena como fecha, no hay necesidad de convertir
    $vDate:=$myObj.myDate
Else  //es almacenado como cadena
    $vDate:=Date($myObj.myDate)
End if
```

Day number

Day number (fecha) -> Resultado

Parámetro	Tipo		Descripción
fecha	Fecha	➡	Fecha para la cual devolver el número del día
Resultado	Entero largo	➲	Número que representa el día de la semana que corresponde a la fecha

Descripción

El comando ***Day number*** *devuelve un número que representa el día de la semana que corresponde a la fecha.*
Nota: Day Number *devuelve 2 para fechas nulas.*
4D ofrece las siguientes constantes predefinidas:

Constante	Tipo	Valor
Sunday	Entero largo	1
Monday	Entero largo	2
Tuesday	Entero largo	3
Wednesday	Entero largo	4
Thursday	Entero largo	5
Friday	Entero largo	6
Saturday	Entero largo	7

Nota: Day number *devuelve un valor entre 1 y 7. Para obtener el número de día en el mes para una fecha, utilice el comando* ***Day of****.*

Ejemplo

El siguiente ejemplo es una función que devuelve el día actual como una cadena:

```
$viDia :=Day number(Current date) ` $viDia toma el valor del número del día actual
Case of
   :($viDia =1)
      $0:="Domingo"
   :($viDia =2)
      $0:="Lunes"
   :($viDia =3)
      $0:="Martes"
   :($viDia =4)
      $0:="Miércoles"
   :($viDia =5)
      $0:="Jueves"
   :($viDia =6)
      $0:="Viernes"
   :($viDia =7)
      $0:="Sábado"
End case
```

Day of

Day of (fecha) -> Resultado

Parámetro	Tipo		Descripción
fecha	Fecha	➡	Fecha para la cual devolver el día
Resultado	Entero largo	➲	Día del mes de la fecha

Descripción

El comando ***Day of*** *devuelve el día del mes de fecha.*
Nota: Day of *devuelve un valor entre 1 y 31. Para obtener el día de la semana de una fecha, utilice el comando* ***Day number****.*

Ejemplo 1

El siguiente ejemplo muestra el uso de ***Day of****. Los valores se asignan a la variable vResult. Los comentarios describen el valor en vResult:*

```
vResult:=Day of(!25/12/92!) ` vResult recibe el valor 25
vResult:=Day of(Current date) ` vResult toma el valor del día de la fecha actual
```

Ejemplo 2

Ver el ejemplo para el comando ***Current date****.*

Milliseconds

Milliseconds -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo	⮌	Número de milisegundos transcurridos desde que se inició el equipo

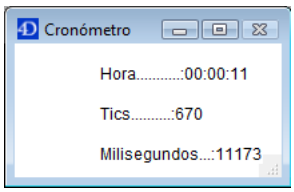
Descripción

Milliseconds *devuelve el número de milisegundos (1milisegundo = 1 milesima de un segundo) pasados desde que se inició el equipo.*

Ejemplo

El siguiente código muestra la ventana "Cronómetro" por un minuto:

```
Open window(100;100;300;200;0;"Cronómetro")
$vhHoraInicio:=Current time
$vlTicsInicio:=Tickcount
$vrMilisegundosInicio:=Milliseconds
Repeat
   GOTO XY(2;1)
   MESSAGE("Hora.....:"+String(Current time-$vhHoraInicio))
   GOTO XY(2;3)
   MESSAGE("Tics.....:"+String(Tickcount-$vlTicsInicio))
   GOTO XY(2;5)
   MESSAGE("Milisegundos...:"+String(Milliseconds-$vrMilisegundosInicio))
Until((Current time-$vhHoraInicio)>=?00:01:00?)
CLOSE WINDOW
```



Month of

Month of (fecha) -> Resultado

Parámetro	Tipo		Descripción
fecha	Fecha	➡	Fecha para la cual devolver el mes
Resultado	Entero largo	⮌	Número que indica el mes de la fecha

Descripción

El comando ***Month of*** *devuelve el mes de fecha.*

Nota: Month of *devuelve el número del mes, no el nombre. (ver ejemplo 1).*

Para comparar el valor devuelto por esta función, 4D ofrece las siguientes constantes predefinidas, que se encuentran en el tema "Days and Months":

Constante	Tipo	Valor
January	Entero largo	1
February	Entero largo	2
March	Entero largo	3
April	Entero largo	4
May	Entero largo	5
June	Entero largo	6
July	Entero largo	7
August	Entero largo	8
September	Entero largo	9
October	Entero largo	10
November	Entero largo	11
December	Entero largo	12

Ejemplo 1

El siguiente ejemplo muestra el uso de ***Month of****. Los resultados se asignan a la variable vResult. Los comentarios describen lo que está en vResult:*

```
vResult:=Month of(!25/12/92!) ` vResult obtiene el valor 12
vResult:=Month of(Current date) ` vResult obtiene el mes de la fecha actual
```

Ejemplo 2

Ver el ejemplo del comando ***Current date****.*

SET DEFAULT CENTURY

SET DEFAULT CENTURY (siglo {; añoPivote})

Parámetro	Tipo		Descripción
siglo	Entero largo	➡	Siglo por defecto (menos uno) para la entrada de años con dos dígitos
añoPivote	Entero largo	➡	Año pivote para la entrada de años con dos dígitos

Descripción

*El comando **SET DEFAULT CENTURY** permite especificar el siglo por defecto y el año pivote utilizado por 4D cuando introduce una fecha con sólo dos dígitos para el año.*

El valor del año pivote define la forma como 4D interpretará la entrada de datos de una fecha con un año de dos dígitos:

- *Si el año es mayor o igual al año pivote, 4D utiliza el siglo actual por defecto.*
- *Si el año es menor que el año pivote, 4D utiliza el siguiente siglo (relativo al siglo actual por defecto).*

Por defecto, 4D establece el siglo 20 como el siglo por defecto y utiliza 30 como año pivote. Por ejemplo:

- *25/01/97 significa 25 de enero de 1997.*
- *25/01/30 significa 25 de enero de 1930.*
- *25/01/29 significa 25 de enero de 2029.*
- *25/01/07 significa 25 de enero de 2007.*

*Para cambiar este comportamiento por defecto, ejecute el comando **SET DEFAULT CENTURY**. El efecto del comando es inmediato. Puede pasar únicamente un nuevo siglo por defecto, o un nuevo siglo por defecto y un año pivote.*

Si pasa sólo un nuevo siglo por defecto menos uno en siglo, 4D interpretará todos los años introducidos con dos dígitos como que pertenecen a este siglo.

Por ejemplo, después de llamar:

```
SET DEFAULT CENTURY(20)  ` Fijar el siglo 21 como siglo por defecto
```

- *25/01/97 significa 25 de enero de 2097*
- *25/01/07 significa 25 de enero de 2007*

Adicionalmente, puede especificar el parámetro opcional añoPivote.

Por ejemplo, después de este llamado, en el cual el año pivote es 1995:

```
SET DEFAULT CENTURY(19;95)  ` Fijar el siglo 21 como siglo por defecto si el año es menor que 95
```

- *25/01/97 significa 25 de enero de 1997*
- *25/01/07 significa 25 de enero de, 2007*

***Nota:** este comando afecta únicamente cómo 4D interpreta las fechas introducidas con años de dos dígitos.*

En todos los casos:

- *25/01/1997 significa enero 25, 1997*
- *25/01/2097 significa enero 25, 2097*
- *25/01/1907 significa enero 25, 1907*
- *25/01/2007 significa enero 25, 2007*

Este comando afecta sólo la entrada de datos. No tiene ningún efecto en el almacenamiento de datos, cálculos, etc.

Tickcount

Tickcount -> Resultado

Parámetro	***Tipo***	***Descripción***
Resultado	*Entero largo*	*Número de tics (60avo de un segundo) pasados desde que se inició el equipo*

Descripción

Tickcount *devuelve el número de tics (1 tic = 1 / 60avo de un segundo) pasados desde que se inició el equipo.*
Nota: Tickcount *devuelve un valor de tipo Entero largo.*

Ejemplo

Ver el ejemplo del comando ***Milliseconds****.*

Time

Time (valHora) -> Resultado

Parámetro	Tipo		Descripción
valHora	Cadena, Entero largo	➡	Valor a devolver en forma de hora
Resultado	Hora	⮌	Hora especificada por horaCadena

Descripción

*El comando **Time** devuelve una expresión de tipo Hora equivalente a la hora especificada en el parámetro valHora.*
El parámetro valHora debe contener:

- *una cadena que contenga una hora expresada en uno de los siguientes formatos de hora estándar de 4D correspondientes al lenguaje de su sistema (para mayor información, consulte la descripción del comando **String**).*
- *un entero largo que representa el número de segundos transcurridos desde 00:00:00.*

***Nota:** si la expresión valHora se evalúa como indefinida, **Time** devuelve una hora vacía (00:00:00). Esto es útil cuando se espera que el resultado de una expresión (por ejemplo, un atributo objeto) sea una hora, incluso si puede ser indefinida.*

Ejemplo 1

El siguiente ejemplo muestra una caja de alerta con el mensaje "1:00 P.M. = 13 horas 0 minutos":

```
ALERT("1:00 P.M. = "+String(Time("13:00:00");Hour Min))
```

Ejemplo 2

Puede expresar todo valor numérico como una hora:

```
vTime:=Time(10000)
  //vTime is 02:46:40
vTime2:=Time((60*60)+(20*60)+5200)
  //vTime2 is 02:46:40
```

Time string

Time string (segundos) -> Resultado

Parámetro	Tipo		Descripción
segundos	Entero largo, Hora	→	Segundos desde la media noche
Resultado	Cadena	←	Hora como una cadena en formato 24 horas

Descripción

El comando **Time string** devuelve la cadena de la expresión tipo hora que usted pasó en segundos.
El formato de la cadena es **HH:MM:SS**.
Si pasa un número de segundos superior al número de segundos que hay en un día (86 400), **Time string** sigue añadiendo horas, minutos y segundos. Por ejemplo, **Time string** (86401) devuelve 24:00:01.
Nota: si necesita el formato de la cadena de la expresión de tipo hora en una variedad de formatos, utilice **String**.

Ejemplo

El siguiente muestra una caja de alerta con el mensaje, "46 800 segundos representan 13:00:00."

```
ALERT("46800 segundos representan "+Time string(46800))
```

Timestamp

Timestamp -> Resultado

Parámetro	Tipo		Descripción
Resultado	Cadena		Hora actual devuelta utilizando el formato ISO con milisegundos

Descripción

Timestamp *devuelve la hora UTC actual en formato ISO con milisegundos, es decir, aaaa-MM-ddTHH:mm:ss.SSSZ. Tenga en cuenta que el carácter "Z" indica la zona horaria GMT.*

Cada hora devuelta por ***Timestamp*** *se expresa de acuerdo con la norma ISO 8601. Para más información sobre este estándar, consulte: https://en.wikipedia.org/wiki/ISO_8601*

Note: *esta función no es adecuada para tiempos; Debe utilizar* ***Milliseconds*** *cuando desee medir el tiempo transcurrido.*

Ejemplo

Puede utilizar ***Timestamp*** *en un archivo de historial para saber con precisión donde ocurrieron los eventos. Como se muestra a continuación, es posible que se produzcan varias operaciones durante el mismo segundo:*

```
$vhDocRef:=Append document("TimestampProject.log")
$logWithTimestamp:=Timestamp+Char(Tab)+"Log with timestamp"+Char(Carriage return)
SEND PACKET($vhDocRef;String($logWithTimestamp))
```

Resultado:

```
2016-12-12T13:31:29.477Z   Log with timestamp
2016-12-12T13:31:29.478Z   Connection of user1
2016-12-12T13:31:29.486Z   ERROR - Exception of type 'System exception'
2016-12-12T13:31:29.492Z   Click on button1684
2016-12-12T13:31:29.502Z   [SP_HELP- 1 rows] Command processed
2016-12-12T13:31:29.512Z   [SP_HELP- 5 rows] Result set fetched
```

Year of

Year of (fecha) -> Resultado

Parámetro	Tipo		Descripción
fecha	Fecha	➡	Fecha para la cual devolver el año
Resultado	Entero largo	➲	Número indicando el año de la fecha

Descripción

*El comando **Year of** devuelve el año de fecha.*

Ejemplo 1

*El siguiente ejemplo muestra el uso de **Year of**. Los resultados se asignan a la variable vResult.*

```
vResult:=Year of(!25/12/92!)  ` vResult toma el valor 1992
vResult:=Year of(!25/12/1992!)  ` vResult toma el valor 1992
vResult:=Year of(!25/12/1892!)  ` vResult toma el valor 1892
vResult:=Year of(!25/12/2092!)  ` vResult toma el valor 2092
vResult:=Year of(Current date)  ` vResult toma el valor del año de la fecha actual
```

Ejemplo 2

*Ver el ejemplo del comando **Current date**.*

Formularios

- *Current form name*
- *Form*
- *FORM FIRST PAGE*
- *FORM Get current page*
- *FORM GET ENTRY ORDER*
- *FORM GET HORIZONTAL RESIZING*
- *FORM GET OBJECTS*
- *FORM GET PARAMETER*
- *FORM GET PROPERTIES*
- *FORM GET VERTICAL RESIZING*
- *FORM GOTO PAGE*
- *FORM LAST PAGE*
- *FORM LOAD*
- *FORM NEXT PAGE*
- *FORM PREVIOUS PAGE*
- *FORM SCREENSHOT*
- *FORM SET ENTRY ORDER*
- *FORM SET HORIZONTAL RESIZING*
- *FORM SET INPUT*
- *FORM SET OUTPUT*
- *FORM SET SIZE*
- *FORM SET VERTICAL RESIZING*
- *FORM UNLOAD*

Current form name

Current form name -> Resultado

Parámetro	Tipo		Descripción
Resultado	Texto		Nombre del formulario de proyecto actual o formulario tabla actual en el proceso

Descripción

*El comando **Current form name** devuelve el nombre del formulario actual definido para el proceso. El formulario actual puede ser un formulario proyecto o un formulario tabla.*

*Por defecto, si no se ha llamado al comando **FORM LOAD** en el proceso actual, el formulario actual es el que está siendo visualizado o impreso. Si ha llamado al comando **FORM LOAD** en el proceso, el formulario actual es el definido para este comando y permanece así hasta que llame a **FORM UNLOAD** (o **CLOSE PRINTING JOB**).*

El comando devuelve:

- *el nombre del formulario, o*
- *el nombre del archivo sin la extensión si el formulario actual es creado por un archivo .json, o*
- *el atributo "nombre" si el formulario actual es creado por un objeto, o*
- *una cadena vacía si no hay una forma actual definida para el proceso.*

Ejemplo 1

En un formulario de entrada, ponga el siguiente código en un bóton:

```
C_TEXT($FormName)
$win:=Open form window([Members];"Input";Plain form window)
DIALOG([Members];"Input")
$FormName:=Current form name
 // $FormName = "Input"
FORM LOAD([Members];"Drag")
$FormName:=Current form name
 // $FormName = "Drag"
 //...
```

Ejemplo 2

Usted quiere obtener el formulario actual si éste es un formulario proyecto:

```
$PointerTable:=Current form table
If(Nil($PointerTable)) // este es un formulario proyecto
   $FormName:=Current form name
   ... // procesamiento
End if
```

Form

Form -> Resultado

Parámetro	Tipo	Descripción
Resultado	Objeto	Datos del formulario asociados al formulario actual

Descripción

*El comando **Form** devuelve el objeto asociado con el formulario actual, si lo hay. 4D asocia automáticamente un objeto al formulario actual en los siguientes casos:*

- *el formulario actual ha sido mostrado por el comando **DIALOG**,*
- *el formulario actual es un subformulario.*

Formulario DIALOG

*Si el formulario actual se muestra mediante una llamada al comando **DIALOG**, **Form** devuelve el objeto formData pasado como parámetro a este comando.*
*Si se omitió el parámetro formData, **Form** devuelve el objeto por defecto creado por **DIALOG**.*

Subformulario

Si el formulario actual es un subformulario, el objeto devuelto depende de la variable del contenedor padre:

- *Si la variable asociada al contenedor principal se ha escrito como un objeto (**C_OBJECT**), **Form** devuelve el valor de esta variable.*
  *En este caso, el objeto devuelto por **Form** es el mismo que el devuelto por la siguiente expresión:*

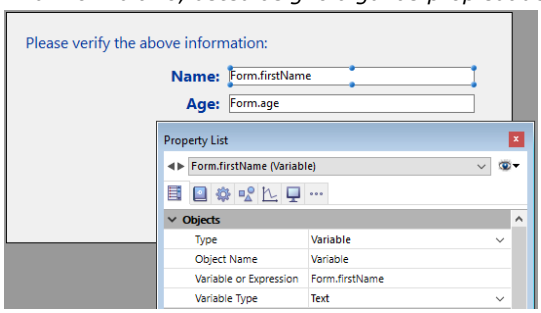
```
(OBJECT Get pointer(Object subform container))->
```

- *Si la variable asociada al contenedor padre no se ha escrito como un objeto, **Form** devuelve un objeto creado automáticamente, mantenido por 4D en el contexto del subformulario.*

*Para más información, consulte la sección **Subformularios en página**.*

Ejemplo

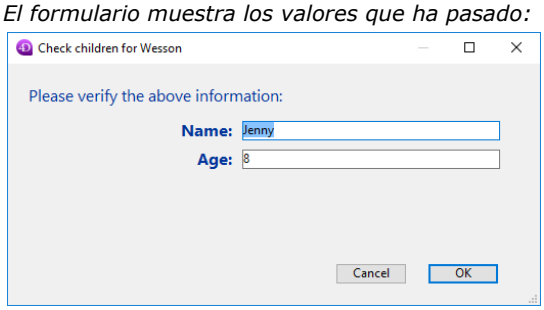
*En un formulario, usted asignó algunas propiedades de objeto **Form** a variables:*



Luego, puede ejecutarlas desde cualquier lugar de la aplicación:

```
C_LONGINT($win)
$win:=Open form window("Edit_Address";Movable form dialog box;Horizontally centered;Vertically centered)
DIALOG("Edit_Address";New object("firstName";"Mike";"age";12))
CLOSE WINDOW($win)
```

El formulario muestra los valores que ha pasado:



***Nota:** este ejemplo requiere que la notación de objeto esté habilitada en la base de datos (ver **Página Compatibilidad**).*

FORM FIRST PAGE

FORM FIRST PAGE
Este comando no requiere parámetros

Descripción

FORM FIRST PAGE *cambia la página actual del formulario por la primera página del formulario.Si ningún formulario es mostrado o cargado por el comando* ***FORM LOAD****, o si la primera página del formulario ya se muestra,* ***FORM FIRST PAGE*** *no hace nada.*

Ejemplo

El siguiente ejemplo es un método de una línea, llamado por un comando de menú, el cual muestra la primera página de un formulario.

```
FORM FIRST PAGE
```

FORM Get current page

FORM Get current page {(*)} -> Resultado

Parámetro	Tipo		Descripción
*	Operador	⇒	Devuelve el número de página de subformulario actual
Resultado	Entero largo	⇐	Número de la página del formulario mostrada actualmente

Descripción

*El comando **FORM Get current page** devuelve el número de la página actual del formulario mostrada actualmente o del formulario actual cargado vía el comando **FORM LOAD**.*

*El parámetro * es útil cuando el comando se llama en el contexto de un subformulario en página que contiene varias páginas. En este caso, cuando pase este parámetro, el comando cambia la página del subformulario actual (la que llamó al comando). Por defecto, cuando se omite el parámetro *, el comando se aplica siempre al formulario padre.*

Ejemplo

In a form, when you select a menu item from the menu bar or when the form receives a call from another process, you can perform different actions depending on the form page currently displayed. In this example, you write:

En un formulario, si selecciona un elemento de menú de la barra de menús o si el formulario recibe una llamada de otro proceso, puede realizar diferentes acciones dependiendo de la página del formulario mostrada actualmente. En este ejemplo, usted escribe:

```
  ` Método de formulario [miTabla];"miForm"
Case of
   :(Form event=On Load)
` ...
   :(Form event=On Unload)
` ...
   :(Form event=On Menu Selected)
      $vlNumeroMenu:=Menu selected>>16
      $vlNumeroCmdo:=Menu selected  & 0xFFFF
      Case of
         :($vlNumeroMenu=...)
            Case of
               :($vlNumeroCmdo=...)
               :(FORM Get current page=1)
` Efectuar una acción apropiada para la página 1
               :(FORM Get current page=2)
` Efectuar una acción apropiada para la página 2
` ...
               :($vlItemNumber=...)
` ...
            End case
         :($vlMenuNumber=...)
` ...
      End case
   :(Form event=On Outside Call)
      Case of
         :(FORM Get current page=1)
` Dar una respuesta apropiada para la página 1
         :(FORM Get current page=2)
` Dar una respuesta apropiada para la página 2
      End case
` ...
End case
```

FORM GET ENTRY ORDER

FORM GET ENTRY ORDER (nomObjetos {; numPag | * })

Parámetro	Tipo		Descripción
nomObjetos	Array texto	⇐	Nombres de objetos ordenados por orden de entrada
numPag | *	Entero largo, Operador	⇒	Número de la página para la que se obtiene el orden de entrada definido (página actual si se omite), o * para obtener el orden de entrada real de la página actual

Descripción

El comando ***FORM GET ENTRY ORDER*** *devuelve en nomObjetos los nombres ordenados de los objetos que definen el orden de entrada del formulario.*

- *Si no pasa el parámetro *,* ***FORM GET ENTRY ORDER*** *devuelve el orden de entrada como se declaró anteriormente con el comando* ***FORM SET ENTRY ORDER****. Puede omitir o pasar el parámetro numPag:*
  - *Si omite el parámetro numPag, el array nomObjetos devuelve el orden de entrada para la página actual,*
  - *Si pasa el parámetro numPag, el array nomObjetos devuelve el orden de entrada para la página numPag.*

  En ambos casos, si el comando ***FORM SET ENTRY ORDER*** *no fue llamado previamente para el formulario actual, el array nomObjetos se devuelve vacío.*

- *Si pasa el parámetro *,* ***FORM GET ENTRY ORDER*** *devuelve el orden de entrada actual de la página actual, es decir, el array nomObjetos sólo contiene nombres de objeto* ***válidos*** *((para más información sobre objetos válidos, consulte la descripción del comando* ***FORM SET ENTRY ORDER****). El orden de entrada de formulario real puede ser:*
  - *El orden de entrada de formulario predeterminado, basado en la superposición de objetos,*
  - *O el orden de entrada del editor de formularios (ver* ***Modificar el orden de entrada de los datos****), si se ha utilizado,*
  - *O el orden de entrada definido por una llamada al comando* ***FORM SET ENTRY ORDER*** *en el proceso actual, si se ha utilizado.*

  El orden de entrada real siempre incluye objetos de la página 0 y de los formularios heredados.

Nota: *el orden de entrada dentro de un subformulario no se devuelve cuando se aplica este comando al formulario padre.*

Ejemplo

Puede excluir ciertos objetos del orden de entrada:

```
ARRAY TEXT($arrTabOrderObject;0)
C_LONGINT($vElem)

FORM GET ENTRY ORDER($arrTabOrderObject;*) //obtener el orden de entrada actual
Repeat
   $vElem:=Find in array($arrTabOrderObject;"vTax@")
   If($vElem>0) //excluye objetos cuyo nombre comienza por "vTax" del orden de entrada de datos
      DELETE FROM ARRAY($arrTabOrderObject;$vElem)
   End if
Until($vElem<0)
FORM SET ENTRY ORDER($arrTabOrderObject) //aplica el nuevo orden de entrada
```

FORM GET HORIZONTAL RESIZING

FORM GET HORIZONTAL RESIZING (redimension {; anchoMin {; anchoMax}})

Parámetro	Tipo		Descripción
redimension	Booleano	⇐	True: el formulario es redimensionable horizontalmente False: el formulario no es redimensionable horizontalmente
anchoMin	Entero largo	⇐	Ancho mínimo del formulario (píxeles)
anchoMax	Entero largo	⇐	Ancho máximo del formulario (píxeles)

Descripción

*El comando **FORM GET HORIZONTAL RESIZING** devuelve las propiedades de redimensionamiento horizontal del formulario actual en las variables redimension, anchoMin y anchoMax. Estas propiedades pueden haberse definido para el formulario en el editor de formularios en modo Diseño o para el proceso actual vía el comando **FORM SET HORIZONTAL RESIZING**.*

FORM GET OBJECTS

FORM GET OBJECTS (arrObjetos {; arrVariables {; arrPags}} {; opcionPag})

Parámetro	Tipo		Descripción
arrObjetos	Array cadena	⇐	Nombre de los objetos del formulario
arrVariables	Array puntero	⇐	Punteros a variables o campos asociados a los objetos
arrPags	Array entero	⇐	Número de página de cada objeto
opcionPag	Entero largo, Operador	⇒	1=Página actual del formulario, 2=Todas las páginas, 4=Páginas heredadas Si se pasa * (obsoleto) = página actual con objetos heredados

Descripción

*El comando **FORM GET OBJECTS** devuelve en forma de array(s) la lista de todos los objetos presentes en el formulario actual. Esta lista puede estar restringida a la página actual del formulario y puede excluir los objetos de los formularios heredados. El comando puede ser utilizado con los formularios de entrada y de salida.*

Si un array pasado como parámetro no ha sido declarado previamente, el comando lo crea y dimensiona automáticamente. Sin embargo, pensando en la compilación de la aplicación, le recomendamos declarar explícitamente cada array.

Pase en arrObjetos el nombre del array alfa que contendrá los nombres de los objetos (cada nombre de objeto es único en un formulario). El orden en el cual los objetos aparecen en el array no es significativo.

Los otros arrays llenados opcionalmente por el comando son sincronizados con el primer array.

*Pase en el parámetro opcional arrVariables el nombre del array puntero que contiene los punteros a las variables o campos asociados a los objetos. Si un objeto no tiene una variable asociada, el puntero **Nil** es devuelto. Si hay un objeto de tipo "subformulario", se devuelve un puntero a la tabla del subformulario.*

El tercer array (opcional), arrPags, se llena con los números de páginas del formulario. Cada línea de este array contiene el número de página del objeto correspondiente.

*El parámetro opcional * le permite reducir la lista de objetos devueltos en la página actual del formulario. Cuando se pasa este parámetro, sólo los objetos de la página actual, de la página 0 y de las páginas heredadas son devueltos por el comando. En otras palabras, todos los objetos presentes en la página actual del formulario (visibles o no) son procesados por el comando.*

*El parámetro opcional opcionPag permite designar la(s) parte(s) del formulario desde donde desea obtener los objetos. Por defecto, si el parámetro opcionPag se omite (así como el parámetro *), se devuelven los objetos de todas las páginas, incluyendo los objetos heredados. Para reducir el alcance del comando, puede pasar un valor en opcionPag. Puede pasar una (o una combinación) de las siguientes constantes, que se encuentran en el tema "**Objetos de formulario (Acceso)**":*

Constante	Tipo	Valor	Comentario
Form all pages	Entero largo	2	Devuelve todos los objetos de todas las páginas, excluyendo los objetos heredados
Form current page	Entero largo	1	Devuelve todos los objetos de la página actual, incluyendo la página 0, pero excluyendo los objetos heredados
Form inherited	Entero largo	4	Devuelve sólo los objetos heredados

***Nota de compatibilidad**: pasar el parámetro * es equivalente a pasar Form current page+Form inherited. La sintaxis con el parámetro * ahora es obsoleta y no debe ser utilizarse más.*

Ejemplo 1

Usted quiere recibir información sobre todas las páginas, incluyendo objetos del formulario heredado (si los hay):

```
//Formulario abierto
FORM GET OBJECTS(arrayObjetos;arrayVariables;arrayPaginas)
```

O:

```
//Formulario cargado
FORM LOAD([Tabla1];"MiForm")
FORM GET OBJECTS(arrayObjetos;arrayVariables;arrayPaginas;Form all pages+Form inherited)
```

Ejemplo 2

Usted desea obtener los objetos de la página actual del formulario cargado, incluyendo la página 0 del formulario y los objetos de los formularios heredados (si los hay):

```
FORM LOAD("MiForm")
FORM GOTO PAGE(2)
FORM GET OBJECTS(arrayObjetos;arrayVariables;arrayPag;Formulario página actual+Formulario heredado)
```

Ejemplo 3

Quiere obtener información sobre todos los objetos en el formulario heredado (si los hay). Si no hay formularios heredados, los arrays se devolverán vacíos.

```
FORM LOAD("MiForm")
FORM GET OBJECTS(arrayObjetos;arrayVariables;arrayPag;Formulario heredado)
```

Ejemplo 4

Usted quiere obtener los objetos de la página 4, incluyendo los de la página 0, pero sin los objetos de formularios heredados (si los hay):

```
FORM LOAD([Tabla1];"MiForm")
FORM GOTO PAGE(4)
FORM GET OBJECTS(arrayObjetos;arrayVariables;arrayPag;Formulario página actual)
```

Ejemplo 5

Usted quiere obtener información los objetos de todas las páginas, pero sin objetos de formulario heredado (si los hubiera):

```
FORM LOAD([Tabla1];"MiForm")
FORM GET OBJECTS(arrayObjetos;arrayVariables;arrayPaginas;Form todas las páginas)
```

FORM GET PARAMETER

FORM GET PARAMETER ({tabla ;} form ; selector ; valor)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del formulario o Tabla por defecto si se omite este parámetro
form	Cadena	➡	Nombre del formulario
selector	Entero largo	➡	Código del parámetro
valor	Entero largo	⬅	Valor actual del parámetro

Descripción

El comando **FORM GET PARAMETER** puede utilizarse para obtener el valor actual de un parámetro del formulario indicado por tabla y form.

selector indica el parámetro del formulario cuyo valor quiere conocer. Puede utilizar la siguiente constante, ubicada en el tema **Parámetro de formulario**:

Constante	Tipo	Valor
NonInverted objects	Entero largo	0

Cuando utiliza la constante **NonInverted Objects** como selector, el comando devuelve, en valor, el modo de visualización real del formulario en modo Menús personalizados bajo Windows. Este parámetro se utiliza en el despliegue de aplicaciones en idiomas "de derecha a izquierda". Para mayor información sobre el soporte de idiomas de derecha a izquierda, por favor consulte el Manual de Diseño de 4D.

- Si valor devuelve 0, los objetos de formulario se invierten,
- Si valor devuelve 1, los objetos de formulario no se invierten.

Si el comando no se llama dentro del contexto del modo Aplicación bajo Windows, siempre devuelve 1.

Recuerde que la inversión efectiva de los objetos de un formulario depende de la combinación de varios parámetros: los valores de la preferencia "Inversión de objetos en modo Aplicación", el valor de la opción de formulario "No invertir objetos" y el sistema en el cual se ejecuta la base de datos. La siguiente tabla especifica el valor devuelto por el comando **FORM GET PARAMETER** dependiendo de las diferentes combinaciones de estos parámetros:

Preferencias: "Inversión de objetos en modo Aplicación"	Propiedades del formulario: "No invertir objetos"	Derecha a izquierda Mostrar bajo Windows	Valor devuelto en FORM GET PARAMETER
No	X	X	1
		X	1
	X		1
			1
Automático	X	X	1
		X	0
	X		1
			1
Sí	X	X	1
		X	0
	X		1
			0

(1) Esta preferencia también puede definirse o leerse utilizando los comandos **SET DATABASE PARAMETER** y **Get database parameter**.

FORM GET PROPERTIES

FORM GET PROPERTIES ({tabla ;} nomForm ; ancho ; alto {; numPags {; largoFijo {; altFijo {; titulo}}}})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del formulario o tabla por defecto, si se omite
nomForm	Cadena	➡	Nombre del formulario
ancho	Entero largo	⬅	Ancho del formulario (en píxeles)
alto	Entero largo	⬅	Altura del formulario (en píxeles)
numPags	Entero largo	⬅	Número de páginas en el formulario
largoFijo	Booleano	⬅	Verdadero = Ancho fijo, Falso = Ancho variable
altFijo	Booleano	⬅	Verdadero = Altura fija, Falso = Altura variable
titulo	Texto	⬅	Título de la ventana del formulario

Descripción

El comando **[#current_title** devuelve las propiedades del formulario nomForm.

Los parámetros ancho y altura devuelven el ancho y la altura del formulario en píxeles. Estos valores son determinados en las propiedades de tamaño de la ventana por defecto del formulario:

- Si el tamaño del formulario es **automático**, su ancho y altura se calculan de manera que todos los objetos del formulario sean visibles, teniendo en cuenta, las márgenes horizontal y vertical que fueron definidas.
- Si el tamaño del formulario es **fijo**, su ancho y altura son introducidos manualmente en las áreas correspondientes.
- Si el tamaño del formulario está **basado en un objeto,** su ancho y altura son calculados con relación a la posición de este objeto.

El parámetro numPags devuelve el número de páginas en un formulario, excluyendo la página 0 (cero).

Los parámetros largoFijo y altoFijo indican si el largo y el alto del formulario son redimensionables (el parámetro devuelve **False**) o fijos (el parámetro devuelve **True**).

El parámetro título devuelve el título de la ventana del formulario, tal como fue definido en la Lista de propiedades del editor de formularios. Si no se definió ningún nombre, el parámetro título devuelve una cadena vacía.

FORM GET VERTICAL RESIZING

FORM GET VERTICAL RESIZING (redimension {; alturaMin {; alturaMax}})

Parámetro	Tipo		Descripción
redimension	Booleano	⇐	True: el formulario es redimensionable verticalmente False: el formulario no es redimensionable verticalmente
alturaMin	Entero largo	⇐	Altura mínima del formulario (píxeles)
alturaMax	Entero largo	⇐	Altura máxima del formulario (píxeles)

Descripción

*El comando **FORM GET VERTICAL RESIZING** devuelve las propiedades de redimensionamiento vertical del formulario actual en las variables redimension, alturaMin y alturaMax. Estas propiedades pueden haberse definido para el formulario en el editor de formularios en modo Diseño o para el proceso actual vía el comando **FORM SET VERTICAL RESIZING**.*

FORM GOTO PAGE

FORM GOTO PAGE (numPag {; *})

Parámetro	Tipo		Descripción
numPag	Entero largo	➡	Número de la página a mostrar
*	Operador	➡	Cambia la página del subformulario actual

Descripción

FORM GOTO PAGE cambia la página actual de un formulario para mostrar la página especificada por numPag.

Si ningún formulario es mostrado o cargado por el comando **FORM LOAD** o si numPag corresponde a la página actual del formulario, **FORM GOTO PAGE** no hace nada. Si numPag es superior que el número de páginas del formulario, se muestra la última página. Si numPag es menor que uno, se muestra la primera página.

El parámetro * es útil cuando el comando se llama en el contexto de un subformulario en página que contiene varias páginas. En este caso, cuando pase este parámetro, el comando cambia la página del subformulario actual (el que llamó al comando). Por defecto, si se omite el parámetro *, el comando se aplica siempre al formulario padre.

Acerca de los comandos de gestión de páginas

Los botones de acción automática realizan las mismas tareas que los comandos **FORM FIRST PAGE**, **FORM LAST PAGE**, **FORM NEXT PAGE**, **FORM PREVIOUS PAGE** y **FORM GOTO PAGE** que puede asociar a los objetos tales como pestañas, list box desplegables, etc. Siempre que sea posible, utilice botones de acción automática en lugar de los comandos.

Los comandos de gestión de página pueden utilizarse con formularios de entrada o con formularios de salida en cajas de diálogo. Los formularios de salida utilizan sólo la primera página. Un formulario siempre tiene por lo menos una página, la primera página. Recuerde que sin importar el número de páginas que tenga un formulario, sólo existe un método de formulario para cada formulario.

- Utilice el comando **FORM Get current page** para saber que página se está mostrando.
- Utilice **Evento formulario** On Page Change que se genera cada vez que la página actual del formulario cambia.

Nota: cuando diseñaun formulario, puede trabajar con las páginas de la 1 a la X, como también con la página 0, en la cual pone los objetos que aparecen en todas las páginas. Cuando **utiliza** un formulario, y llama los comandos de página, usted trabaja con las páginas de la 1 a la X; la página 0 se combina automáticamente con la página que está siendo mostrada.

Ejemplo

El siguiente ejemplo es un método de objeto para un botón que muestra una página específica, la página 3:

```
FORM GOTO PAGE(3)
```

FORM LAST PAGE

FORM LAST PAGE
Este comando no requiere parámetros

Descripción

FORM LAST PAGE *cambia la página actual de un formulario para mostrar la última página del formulario. Si ningún formulario es mostrado o cargado por el comando* ***FORM LOAD****, o si ya se muestra la última página del formulario,* ***FORM LAST PAGE*** *no hace nada.*

Ejemplo

El siguiente ejemplo es un método de una línea, llamado por un comando de menú, el cual muestra la última página del formulario:

```
FORM LAST PAGE
```

FORM LOAD

FORM LOAD ({aTabla ;} formulario {; *})

Parámetro	Tipo		Descripción
aTabla	Tabla	➡	Tabla del formulario a cargar (si se omite, carga un formulario proyecto)
formulario	Cadena, Objeto	➡	Nombre del formulario proyecto a abrir para la impresión o Cadena vacía para cerrar el formulario proyecto actual
*	Operador	➡	Si se pasa = el comando se aplica a la base local cuando se ejecuta desde un componente (parámetro ignorado fuera de este contexto)

Descripción

*El comando **FORM LOAD** se utiliza para cargar el formulario en memoria en el proceso actual con el fin de imprimir sus datos o analizar sus contenidos. Sólo puede haber un formulario actual por proceso.*

En el parámetro formulario, puede pasar:

- *el nombre de un formulario, o*
- *la ruta (en sintaxis POSIX) a un archivo .json válido que contiene una descripción del formulario a usar (ver **Ruta de archivo del formulario**), o*
- *un objeto que contiene una descripción del formulario.*

Impresión de datos

*Para que este comando pueda ejecutarse, una tarea de impresión debe haberse abierto de antemano usando el comando **OPEN PRINTING JOB**. El comando **OPEN PRINTING JOB** hace un llamado implícito al comando **FORM UNLOAD**, por lo que en este contexto es necesario ejecutar **FORM LOAD**. Una vez cargado, el formulario se convierte en el formulario de impresión actual. Todos los comandos de gestión de objetos, y en particular, el comando **Print object**, trabajan con este formulario.*

*Si un formulario de impresión ya se ha cargado previamente (a través de una llamada anterior al comando **FORM LOAD**), se cierra y se sustituye por formulario. Puede abrir y cerrar varios formularios proyecto en la misma sesión de impresión. Cambiar de formulario de impresión vía el comando **FORM LOAD** no genera saltos de página. Es responsabilidad del desarrollador gestionar los saltos de página.*

Sólo el evento formulario On Load se ejecuta durante la apertura del formulario, así como los métodos de los objetos del formulario. Se ignoran los otros eventos formulario. El evento de formulario On Unload se ejecuta al final de la impresión.

Para mantener la coherencia gráfica de los formularios, se recomienda aplicar la propiedad de apariencia "Impresión ", independientemente de la plataforma.

*El formulario de impresión actual se cierra automáticamente cuando se llama el comando **CLOSE PRINTING JOB**.*

***Nota de compatibilidad**: en las versiones de 4D anteriores a la v14, el comando **FORM LOAD** (llamado OPEN PRINTING FORM) aceptaba una cadena vacía en el parámetro formulario para cerrar el formulario de proyecto actual. Esta sintaxis ya no se admite y devuelve un error. Debe utilizar el comando **FORM UNLOAD** o el comando **CLOSE PRINTING JOB** para cerrar el formulario.*

Análisis del contenido del formulario

*Esta posibilidad consiste en cargar un formulario fuera de pantalla para análisis. Para efectuar esta acción, basta con llamar a **FORM LOAD** fuera del contexto de un trabajo de impresión. En este caso, los eventos de formulario no se ejecutan.*

***FORM LOAD**] se puede utilizar con los comandos **FORM GET OBJECTS** y **OBJECT Get type** para llevar a cabo cualquier tipo de procesamiento en el contenido del formulario. A continuación, es imperativo llamar al comando **FORM UNLOAD** para descargar el formulario de la memoria.*

*Tenga en cuenta que en todos los casos, el formulario en la pantalla permanece cargado (no se ve afectado por el comando **FORM LOAD** por lo que no es necesario volver a cargarlo después de llamar a **FORM UNLOAD**.*

*Cuando el comando se ejecuta desde un componente, carga los formularios componente por defecto. Si pasa el parámetro *, el método carga los formularios de la base local.*

***Recordatorio:** en el contexto fuera de pantalla, no olvide llamar **FORM UNLOAD** para evitar todo riesgo de saturación de la memoria.*

Ejemplo 1

Llamar un formulario proyecto en un trabajo de impresión:

```
OPEN PRINTING JOB
FORM LOAD("print_form")
 // ejecución de eventos y métodos objeto
```

Ejemplo 2

Llamar un formulario tabla en un trabajo de impresión:

```
OPEN PRINTING JOB
FORM LOAD([People];"print_form")
 // ejecución de eventos y métodos de objeto
```

Ejemplo 3

Análisis del contenido de un formulario para efectuar un procesamiento en las áreas de entrada de texto:

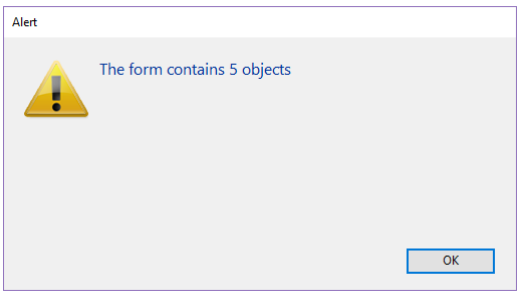
```
FORM LOAD([People];"my_form")
 // selección del formulario sin ejecución de los eventos ni de los métodos
FORM GET OBJECTS(arrObjNames;arrObjPtrs;arrPages;*)
For($i;1;Size of array(arrObjNames))
   If(OBJECT Get type(*;arrObjNames{$i})=Object type text input)
 //··· procesamiento
   End if
End for
FORM UNLOAD  //no olvidar descargar el formulario
```

Ejemplo 4

The following example returns the number of objects on a JSON form:

```
ARRAY TEXT(objectsArray;0)  //sort form items into arrays
ARRAY POINTER(variablesArray;0)
ARRAY INTEGER(pagesArray;0)

FORM LOAD("/RESOURCES/OutputForm.json")  //load the form
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form all pages+Form inherited)

ALERT("The form contains "+String(size of array(objectsArray))+" objects")  //return the object count
```

the result shown is:



FORM NEXT PAGE

FORM NEXT PAGE
Este comando no requiere parámetros

Descripción

FORM NEXT PAGE *cambia la página actual del formulario para mostrar la página siguiente.Si ningún formulario es mostrado o cargado por el comando* ***FORM LOAD****, o si ya se muestra la última página del formulario,* ***FORM NEXT PAGE*** *no hace nada.*

Ejemplo

El siguiente ejemplo es un método de una línea, llamado por un comando de menú, el cual muestra la página del formulario que sigue a la página mostrada actualmente:

```
FORM NEXT PAGE
```

FORM PREVIOUS PAGE

FORM PREVIOUS PAGE
Este comando no requiere parámetros

Descripción

FORM PREVIOUS PAGE *cambia la página actual de un formulario para mostrar la página anterior. Si ningún formulario es mostrado o cargado por el comando* ***FORM LOAD*** *o si ya se muestra la primera página del formulario,* ***FORM PREVIOUS PAGE*** *no hace nada.*

Ejemplo

El siguiente ejemplo es un método de una línea llamado por un comando de menú, el cual muestra la página del formulario anterior a la página mostrada actualmente:

```
FORM PREVIOUS PAGE
```

FORM SCREENSHOT

FORM SCREENSHOT ({{tabla ;} nomForm ;} imagForm {; pagNum})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del formulario
nomForm	Texto	➡	Nombre del formulario
imagForm	Imagen	⬅	Imagen del formulario en ejecución si el primer parámetro se omite, o Imagen del formulario en el editor de formularios si se pasa un nombre de formulario
pagNum	Entero largo	➡	Número de página del formulario

Descripción

*El comando **FORM SCREENSHOT** devuelve un formulario en forma de imagen. Este comando admite dos sintaxis diferentes: en función de la sintaxis utilizada, puede obtener la imagen de un formulario ejecutado, o la imagen del formulario en el editor de formularios.*

- *FORM SCREENSHOT (imagForm)*
  *Esta sintaxis permite obtener una captura de pantalla de la página actual del formulario en ejecución o cargado vía el comando **FORM LOAD**: la imagen devuelta en el parámetro imagenForm contiene todos los objetos visibles del formulario con los valores actuales de los campos y de las variables del formulario, subformulario, etc. El formulario es devuelto en su totalidad, sin tener en cuenta el tamaño de la ventana que lo contiene.*
  Tenga en cuenta que esta sintaxis sólo funciona con formularios de entrada.

- *FORM SCREENSHOT ({tabla ;} nomForm; imagForm {; pagNum})*
  Esta sintaxis permite obtener una captura de pantalla de una "plantilla" de formulario como la que se muestra en el editor de formularios. Todos los objetos visibles se dibujan como en el editor, el comando tiene en cuenta los formularios heredados y los objetos ubicados en la página 0.
  Si desea una captura de pantalla de un formulario tabla, pase la tabla del formulario en el parámetro tabla y luego su nombre como una cadena en nomForm. Para un formulario proyecto, pase directamente el nombre del formulario en nomForm.
  Por defecto, el comando devuelve una captura de pantalla de la página 1 del formulario. Si sólo desea una imagen de la página 0, o de cualquier otra página del formulario, pase el número de página en el parámetro pagNum.

Notas:

- *Las áreas web no se dibujan en la captura de pantalla devuelta.*
- *Los dos primeros parámetros de este comando son opcionales, no puede pasar directamente como un argumento una función que devuelva un puntero como **Current form table**-> o **Table**->. Aunque esta sintaxis funcionaría en modo interpretado, sería rechazada durante la compilación, así que es necesario en este caso utilizar una variable puntero intermediaria. Para obtener más información, consulte "**Uso directo de los comandos que devuelven punteros**".*

FORM SET ENTRY ORDER

FORM SET ENTRY ORDER (nomObjetos {; numPag})

Parámetro	Tipo		Descripción
nomObjetos	Array texto	➡	Array de nombres de objetos en su orden de entrada esperado
numPag	Entero largo	➡	Número de la página para definir el orden de entrada (página actual si se omite)

Descripción

*El comando **FORM SET ENTRY ORDER** permite definir dinámicamente el orden de entrada del formulario actual para el proceso actual basado en el array nomObjetos.*

Pase en nomObjetos un array que contenga los nombres de los objetos de formulario a incluir en el orden de entrada. El orden de los objetos en el array define el orden de entrada del formulario. Todo objeto de formulario válido perteneciente al formulario actual puede ser listado. Un objeto es válido si:

- *tiene la propiedad enfocable (**Nota**: el comando ignora la propiedad **Tabulable** de los objetos),*
- *existe en el formulario (su nombre está definido),*
- *se utiliza en la página actual (o en la página numPag, ver abajo). Tenga en cuenta que una página de formulario incluye los objetos de la página 0 y los objetos del formulario heredado.*

*Si se detecta un objeto no válido en tiempo de ejecución, simplemente se omite y 4D intentará utilizar el siguiente objeto válido en el array nomObjetos. Puede conocer el orden de entrada actual de la página actual (basada en objetos válidos) utilizando el comando **FORM GET ENTRY ORDER** con el parámetro *.*

Opcionalmente, puede pasar el numPag para el cual definir el orden de entrada. Si se omite, el comando se aplica a la página actual.

Notas:

- *El orden de entrada de un subformulario se define en el propio subformulario. Debe llamar al comando **FORM SET ENTRY ORDER** en el contexto del subformulario.*
- *Este comando no define el primer objeto enfocable en el formulario en tiempo de ejecución. Si desea definir un primer objeto en el orden de entrada, debe utilizar el comando **GOTO OBJECT** en el evento On Load del formulario. Si utilizó el comando **OBJECT DUPLICATE**, puede definir el objeto duplicado como el primero pasando la constante Object First in entry order en el parámetro ligadoA.*

Acerca del orden de entrada de los datos
*El orden de entrada de los datos es el orden en que se seleccionan los campos, subformularios y todos los demás objetos activos cuando el usuario toca la teclae **Tab** o **Retorno de carro** en un formulario. El orden inverso de entrada de datos también está disponible presionando las teclas **Mayús +Tab** o **Mayús +Retorno de carro**. El orden de entrada puede definirse por defecto o modificarse en el editor de formularios. Para más información, consulte la sección **Modificar el orden de entrada de los datos** del manual de Diseño.*

Ejemplo

Usted desea definir el orden de entrada de los objetos en el formulario basado en sus nombres:

```
ARRAY TEXT(tabNames;0)

FORM GET OBJECTS(tabNames;Form current page+Form inherited) //obtenemos los nombres de los objeto del formulario
SORT ARRAY(tabNames;>) //clasifica los nombres en orden ascendente
FORM SET ENTRY ORDER(tabNames) //utiliza el orden alfabético para el orden de entrada
 //se ignoran los objetos no enfocables
```

FORM SET HORIZONTAL RESIZING

FORM SET HORIZONTAL RESIZING (redimension {; anchoMin {; anchoMax}})

Parámetro	Tipo		Descripción
redimension	Booleano	➡	True: el formulario es redimensionable horizontalmente False: El formulario no puede redimensionarse horizontalmente
anchoMin	Entero largo	➡	ancho mínimo del formulario (píxeles)
anchoMax	Entero largo	➡	ancho máximo del formulario (píxeles)

Descripción

*El comando **FORM SET HORIZONTAL RESIZING** permite cambiar por programación las propiedades de redimensionamiento horizontal del formulario actual. Por defecto, estas propiedades son definidas en el editor de formularios en el entorno Diseño. Las nuevas propiedades son definidas para el proceso actual; no se almacenan con el formulario.*

El parámetro redimension le permite definir si el formulario puede redimensionarse horizontalmente; en otras palabras, si el ancho es modificable (manualmente por el usuario o por programación).

*Si pasa **True**, el ancho del formulario puede ser modificado por el usuario; 4D utiliza como marcadores los valores pasados en anchoMin y anchoMax.*

*Si pasa **False**, no se puede modificar el largo del formulario actual; en este caso, no hay necesidad de pasar valores en los parámetros anchoMin y anchoMax .*

*Si ha pasado **True** en el primer parámetro, puede pasar en los parámetros opcionales anchoMin y anchoMax los nuevos largos, mínimos y máximos, del formulario (en píxeles). Si omite estos parámetros se utilizan los valores definidos en el entorno Diseño (si los hay).*

Ejemplo

*Consulte el ejemplo del comando **FORM SET SIZE**.*

FORM SET INPUT

FORM SET INPUT ({tabla ;} form {; formUsuario {; *}})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual definir el formulario de entrada o Tabla por defecto, si se omite
form	Cadena, Objeto	➡	Nombre del formulario a definir como formulario de entrada
formUsuario	Cadena	➡	Nombre del formulario usuario a utilizar
*		➡	Tamaño de ventana automático

Descripción

El comando **FORM SET INPUT** define el formulario de entrada actual de tabla para form o formUsuario. El formulario debe pertenecer a tabla.

El alcance de este comando es el proceso actual. Cada tabla tiene su propio formulario de entrada en cada proceso.

En el parámetro nomForm, puede pasar:

- el nombre de un formulario (formulario proyecto o tabla) a utilizar;
- la ruta (en sintaxis POSIX) a un archivo .json válido que contiene una descripción del formulario a usar. Ver **Ruta de archivo del formulario**;
- un objeto que contiene una descripción del formulario a utilizar.

Nota: por razones estructurales, este comando no es compatible con formularios de proyectos.

FORM SET INPUT no muestra el formulario; sólo designa cuál formulario se utiliza para la entrada de datos, importación, u operación por otro comando. Para mayor información sobre la creación de formularios, consulte el Manual de Diseño 4D.

El formulario de entrada por defecto para cada tabla se define en la ventana del Explorador. Este formulario de entrada por defecto se utiliza si el comando **FORM SET INPUT** no se utiliza para especificar un formulario de entrada, o si especifica un formulario que no existe.

Los formularios de entrada también se utilizan automáticamente mediante acciones estándar como ak edit subrecord o ak add subrecord.

El parámetro opcional formUsuario le permite especificar un formulario usuario (proveniente de form) como formulario de entrada por defecto. Si pasa un nombre de formulario usuario correcto, este formulario será utilizado por defecto en lugar del formulario de entrada en el proceso actual. Esto le permite tener simultáneamente diferentes formularios usuarios personalizados (generados utilizando el comando **CREATE USER FORM**) y utilizar aquel que sea conveniente en función del contexto.

Para mayor información sobre formularios de usuario, consulte la sección **Presentación de los formularios de usuario**.

Los formularios de entrada son mostrados por numerosos comandos, los cuales generalmente se utilizan para permitir al usuario introducir nuevos datos o modificar datos antiguos. Los siguientes comandos muestran un formulario de entrada para entrada de datos o búsquedas:

- **ADD RECORD**
- **DISPLAY RECORD**
- **MODIFY RECORD**
- **QUERY BY EXAMPLE**

Los comandos **DISPLAY SELECTION** y **MODIFY SELECTION** muestran una lista de registros utilizando el formulario de salida. El usuario puede hacer doble clic en un registro en la lista y se muestra el formulario de entrada.

Los comandos de importación **IMPORT TEXT**, **IMPORT SYLK** e **IMPORT DIF** utilizan el formulario de entrada actual para importar registros.

El parámetro opcional * se utiliza en conjunto con las propiedades del formulario que definió en la ventana de propiedades del formulario del entorno Diseño y el comando **Open window**. El especificar el parámetro * le indica a 4D que utilice las propiedades del formulario para redimensionar automáticamente la ventana para el siguiente uso del formulario (como un formulario de entrada o como una caja de diálogo). Ver mayor información en **Open window**.

Nota: bien sea que pase el parámetro opcional * o no, **FORM SET INPUT** cambia el formulario de entrada para la tabla.

Ejemplo 1

El siguiente ejemplo muestra un uso típico de **FORM SET INPUT**:

```
FORM SET INPUT([Empresas];"Nueva empresa") ` Formulario para añadir nuevas empresas
ADD RECORD([Empresas]) ` Añadir una nueva empresa
```

Ejemplo 2

En una base de facturación que administra varias empresas, la creación de una factura debe efectuarse utilizando el formulario usuario correspondiente:

```
Case of
   :(empresa="4D SAS")
      FORM SET INPUT([Facturas];"Entrada";"4D_SAS")
   :(empresa="4D Inc")
```

```
        FORM SET INPUT([Facturas];"Entrada";"4D_Inc")
    :(empresa="Acme")
        FORM SET INPUT([Facturas];"Entrada";"ACME")
End case
ADD RECORD([Facturas])
```

Ejemplo 3

Los siguientes ejemplos usan la ruta a un formulario .json para ingresar los registros en una lista de empleados:

```
FORM SET INPUT([Personnel];"/RESOURCES/PersonnelForm.json")
ADD RECORD([Personnel])
```

que devuelve:

FORM SET OUTPUT

FORM SET OUTPUT ({tabla ;} form {; formUsuario})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual definir el formulario de salida, o Tabla por defecto, si se omite
form	Cadena, Objeto	➡	Nombre del formulario
formUsuario	Cadena	➡	Nombre del formulario usuario a utilizar

Descripción

*El comando **FORM SET OUTPUT** define el formulario de salida actual de formulario o formUsuario. El formulario debe pertenecer a tabla.*

El parámetro form es el formulario que se mostrará. Pase el:

- *el nombre de un formulario;*
- *la ruta (en sintaxis POSIX) a un archivo .json válido que contiene una descripción del formulario a usar. Ver **Ruta de archivo del formulario**;*
- *un objeto que contiene una descripción del formulario.*

El alcance de este comando es el proceso actual. Cada tabla tiene su propio formulario de salida en cada proceso.

***Nota**: por razones estructurales, este comando no es compatible con formularios de proyecto.*

***FORM SET OUTPUT** no muestra el formulario; simplemente designa el formulario que debe imprimir, mostrar, o utilizar otro comando. Para mayor información sobre la creación de formularios, consulte el Manual de Diseño.*

*El formulario de salida por defecto se define en la ventana del Explorador en el entorno Diseño para cada tabla. Este formulario de salida por defecto se utiliza si el comando **FORM SET OUTPUT** no se utiliza para especificar un formulario de salida, o si usted especifica un formulario que no existe.*

*El parámetro opcional formUsuario le permite especificar un formulario usuario (que viene desde formulario) como formulario de salida por defecto. Si pasa un nombre de formulario de usuario correcto, este formulario será utilizado por defecto en lugar del formulario de salida en el proceso actual. Esto le permite tener simultáneamente diferentes formularios de usuario personalizados (generados utilizando el comando **CREATE USER FORM**) y utilizar el que convenga de acuerdo al contexto.*

*Para mayor información sobre formularios usuario, consulte la sección **Presentación de los formularios de usuario**.*

*Los formularios de salida son utilizados por tres grupos de comandos. Un grupo muestra una lista de registros en pantalla, otro grupo genera informes, y el tercer grupo exporta datos. Los comandos **DISPLAY SELECTION** y **MODIFY SELECTION** muestran una lista de registros utilizando un formulario de salida. Utilice el formulario de salida durante la creación de informes con los comandos **PRINT LABEL** y **PRINT SELECTION** . Cada uno de los comandos de exportación (**EXPORT DIF**, **EXPORT SYLK** y **EXPORT TEXT**) utiliza también el formulario de salida.*

Ejemplo 1

*El siguiente ejemplo muestra un uso típico de **FORM SET OUTPUT**. Note que aunque el comando **FORM SET OUTPUT** aparece inmediatamente antes de que el formulario sea utilizado, no es obligatorio. De hecho, el comando podría ejecutarse en un método completamente diferente, siempre y cuando se ejecute antes de este método:*

```
FORM SET INPUT([Parts];"Parts In") //Selección del formulario de entrada
FORM SET OUTPUT([Parts];"Parts List") //Selección del formulario de salida
MODIFY SELECTION([Parts]) //Este comando utiliza ambos formularios
```

Ejemplo 2

Los siguientes ejemplos usan la ruta a un formulario .json para imprimir los registros en una lista de empleados:

```
FORM SET OUTPUT([Personnel];"/RESOURCES/PersonnelForm.json")
ALL RECORDS([Personnel])
PRINT SELECTION([Personnel])
```

FORM SET SIZE

FORM SET SIZE ({objeto ;} horizontal ; vertical {; *})

Parámetro	Tipo		Descripción
objeto	Cadena	➡	Nombre del objeto que indica los límites del formulario
horizontal	Entero largo	➡	Si se pasa *: margen horizontal (píxeles) Si se omite *: ancho (píxeles)
vertical	Entero largo	➡	Si se pasa *: margen vertical (píxeles) Si se omite *: altura (píxeles)
*	Operador	➡	• Si se pasa: añadir las márgenes definidas por los parámetros horizontal y vertical (tamaño automático o basado en un objeto, si se pasa un objeto) • Si se omite: utilizar horizontal y vertical como ancho y altura del formulario

Descripción

*El comando **FORM SET SIZE** le permite cambiar el tamaño del formulario actual por programación. El nuevo tamaño es definido por el proceso actual; no está almacenado con el formulario.*

Como en el entorno Diseño, puede utilizar este comando para definir el tamaño del formulario de tres maneras:

- *Automáticamente, 4D determina el tamaño del formulario basado en la noción de que todos los objetos deben ser visibles y eventualmente añadiendo una margen horizontal y vertical,*
- *Basado en la ubicación de un objeto del formulario, al cual se añaden eventualmente una margen horizontal y una margen vertical,*
- *Introduciendo tamaños "fijos" (ancho y altura).*

Para mayor información sobre los posibles redimensionamientos de los formularios, consulte el Manual de Diseño de 4D.

- ***Tamaño automático***

Si quiere que el tamaño del formulario se defina de manera automática, debe utilizar la siguiente sintaxis:

```
FORM SET SIZE(horizontal;vertical;*)
```

En este caso, debe pasar las márgenes (en píxeles) que quiere añadir a la derecha y en la parte inferior del formulario en horizontal and vertical.

- ***Tamaño basado en un objeto***

Si quiere que el tamaño del formulario esté basado en un objeto, debe utilizar la siguiente sintaxis:

```
FORM SET SIZE(objeto;horizontal;vertical)
```

*En este caso, debe pasar las márgenes (en píxeles) que quiere añadir a la derecha y en la parte inferior del objeto en horizontal y vertical. No puede pasar el parámetro *.*

- ***Tamaño fijo***

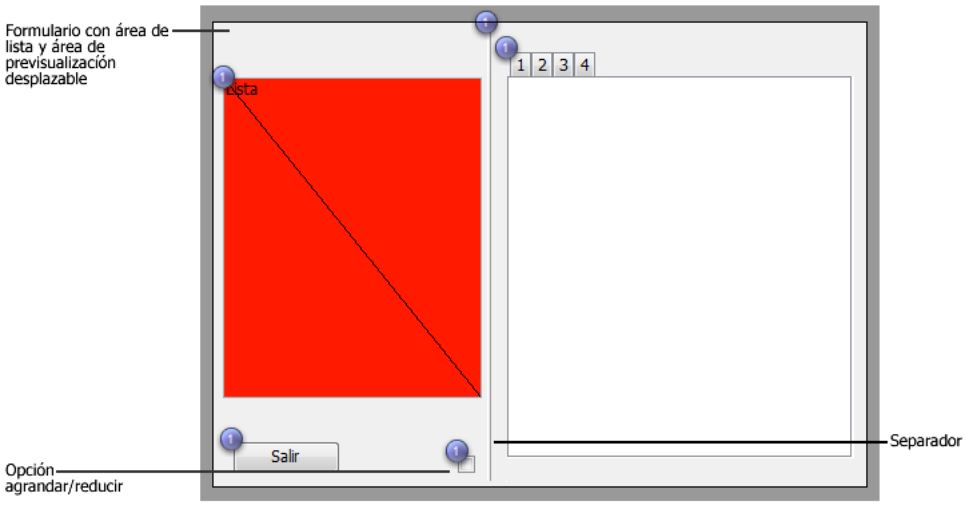
Si quiere tener un tamaño de formulario fijo, debe utilizar la siguiente sintaxis:

```
FORM SET SIZE(horizontal;vertical)
```

En este caso, debe pasar el ancho y la altura (en píxeles) del formulario en horizontal y vertical.
*El comando **FORM SET SIZE** cambia el tamaño del formulario, pero también tiene en cuenta las propiedades de redimensionamiento. Por ejemplo, si el ancho mínimo de un formulario es 500 píxeles y si el comando define un ancho de 400 píxeles, el nuevo ancho del formulario será de 500 píxeles.*
*Igualmente note que este comando no cambia el tamaño de la ventana del formulario (puede redimensionar un formulario sin cambiar el tamaño de la ventana y viceversa). Para cambiar el tamaño de la ventana del formulario, consulte el comando **RESIZE FORM WINDOW**.*

Ejemplo

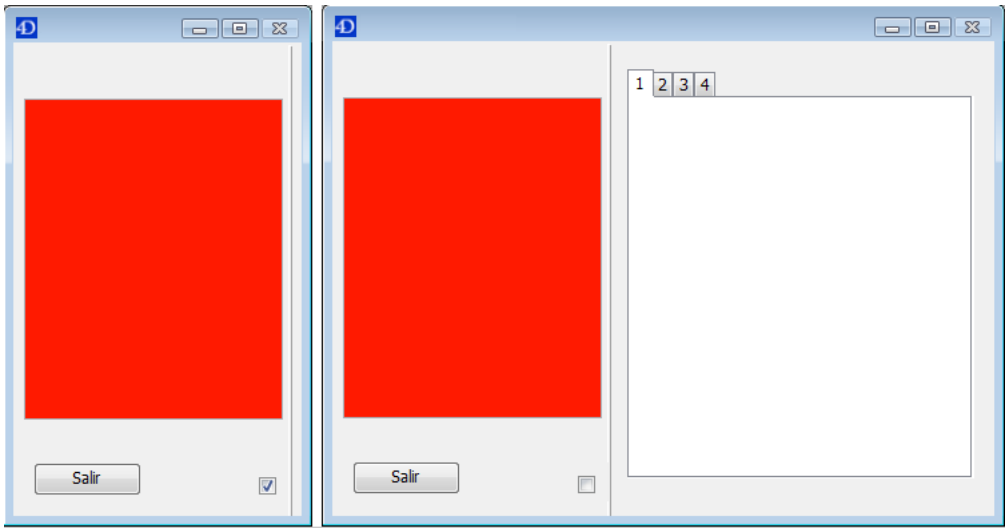
El siguiente ejemplo muestra cómo colocar una ventana de tipo Explorador. El siguiente formulario se crea en el entorno Diseño:



El tamaño del formulario es "automático".
La ventana se visualiza utilizando el siguiente código:

```
$ref:=Open form window([Tabla 1];"Form1";Standard form window;Horizontally centered;Vertically centered;*)
DIALOG([Tabla 1];"Form1")
CLOSE WINDOW
```

La parte de la derecha de la ventana puede mostrarse u ocultarse haciendo clic en la opción agrandar/reducir:



El método de objeto asociado con este botón es el siguiente:

```
Case of
   :(Form event=On Load)
      C_BOOLEAN(b1;<>contraido)
      C_LONGINT(margen)
      margen:=15
      b1:=<>contraido
      If(<>contraido)
         FORM SET HORIZONTAL RESIZING(False)
         FORM SET SIZE("b1";margen;margen)
      Else
         FORM SET HORIZONTAL RESIZING(True)
         FORM SET SIZE("tab";margen;margen)
      End if

   :(Form event=On Clicked)
      <>contraido:=b1
      If(b1)
`contraido
         OBJECT GET COORDINATES(*;"b1";$l;$t;$r;$b)
         GET WINDOW RECT($lf;$tf;$rf;$bf;Current form window)
         SET WINDOW RECT($lf;$tf;$lf+$r+margen;$tf+$b+margen;Current form window)
         FORM SET HORIZONTAL RESIZING(False)
         FORM SET SIZE("b1";margen;margen)

      Else
```

```
    `expandido
            OBJECT GET COORDINATES(*;"tab";$l;$t;$r;$b)
            GET WINDOW RECT($lf;$tf;$rf;$bf;Current form window)
            SET WINDOW RECT($lf;$tf;$lf+$r+margen;$tf+$b+margen;Current form window)
            FORM SET HORIZONTAL RESIZING(True)
            FORM SET SIZE("tab";margen;margen)
        End if

End case
```

FORM SET VERTICAL RESIZING

FORM SET VERTICAL RESIZING (redimension {; alturaMin {; alturaMax}})

Parámetro	Tipo		Descripción
redimension	Booleano	➡	True: el formulario es redimensionable verticalmente False: el formulario no puede ser redimensionado verticalmente
alturaMin	Entero largo	➡	Altura mínima del formulario (píxeles)
alturaMax	Entero largo	➡	Altura máxima del formulario (píxeles)

Descripción

El comando ***FORM SET VERTICAL RESIZING*** *le permite modificar por programación las propiedades de redimensionamiento vertical del formulario actual. Por defecto, estas propiedades son definidas en el editor de formularios en el entorno Diseño. Las nuevas propiedades son definidas para el proceso actual; no son almacenadas con el formulario.*

El parámetro redimension le permite definir si el formulario puede redimensionarse verticalmente; en otras palabras, si la altura es modificable (manualmente por el usuario o por programación).

Si pasa ***True****, la altura del formulario puede ser modificada por el usuario; 4D utiliza como marcadores los valores pasados en alturaMin y alturaMax.*

Si pasa ***False****, no se puede modificar el largo del formulario actual; en este caso, no hay necesidad de pasar valores en los parámetros alturaMin y alturaMax.*

Si ha pasado ***True*** *en el primer parámetro, puede pasar en los parámetros opcionales alturaMin y alturaMax las nuevas alturas, mínimas y máximas, del formulario (en píxeles). Si omite estos parámetros se utilizan los valores definidos en el entorno Diseño (si los hay).*

Ejemplo

Consulte el ejemplo del comando ***FORM SET SIZE****.*

FORM UNLOAD

FORM UNLOAD
Este comando no requiere parámetros

Descripción

*El comando **FORM UNLOAD** libera de la memoria el formulario actual designado utilizando el comando **FORM LOAD**.*
*Llamar este comando es necesario cuando se utiliza el comando **FORM LOAD** fuera del contexto de impresión (en el caso de la impresión, el formulario actual se cierra de nuevo automáticamente cuando se llama el comando **CLOSE PRINTING JOB**).*

Formularios de usuario

Presentación de los formularios de usuario
CREATE USER FORM
DELETE USER FORM
EDIT FORM
LIST USER FORMS

Presentación de los formularios de usuario

En 4D, los desarrolladores pueden ofrecer a los usuarios la posibilidad de crear o modificar formularios personalizados. Estos "Formularios de usuario" pueden ser utilizados como cualquier otro formulario de 4D.

Introducción

Los formularios de usuario están basados en formularios 4D estándar creados por el desarrollador en modo Diseño (llamados formularios "fuente" o "desarrollador") donde se aplica la propiedad ***Modificable por el usuario*** *en el editor de formularios. Un editor de formularios simplificado (llamado utilizando el comando* ***EDIT FORM****) permite a los usuarios modificar la apariencia del formulario, añadir objetos gráficos (utilizando una librería de objetos específicos), ocultar elementos, etc. el desarrollador puede controlar las acciones autorizadas.*

Los formularios de usuario pueden utilizarse de dos maneras diferentes:

- *El usuario modifica el formulario "fuente" para adaptarlo a sus necesidades con la ayuda del comando* ***EDIT FORM****. El formulario de usuario se conserva localmente y se utiliza automáticamente en lugar del formulario original.*

Este funcionamiento responde a las necesidades del desarrollador de definir parámetros en el sitio para cajas de diálogo; por ejemplo, para añadir el logo de la empresa en los formularios, ocultar campos innecesarios, etc.

- *El formulario "fuente" actúa como una plantilla base que los usuarios pueden duplicar libremente y generar tantas copias como lo consideren necesario utilizando el comando* ***CREATE USER FORM****. Es posible definir los parámetros libremente en cada copia (contenido, nombre, etc.) utilizando el comando* ***EDIT FORM****. Sin embargo, el nombre de cada formulario de usuario debe ser único. Los comandos* ***FORM SET INPUT*** *y* ***FORM SET OUTPUT*** *permiten especificar el formulario de usuario a utilizar en cada proceso.*

Este funcionamiento permite a los desarrolladores crear, por ejemplo, informes personalizados.

Guardar y administrar formularios de usuario

Los mecanismos de los formularios de usuario funcionan con las bases compiladas e interpretadas, con 4D en modo local, 4D Server o 4D Desktop. En modo cliente/servidor, los formularios modificados por el usuario están disponibles en todos los equipos.

4D trata automáticamente la gestión de cambios en los formularios. Cuando un formulario está declarado como ***Modificable por el usuario****, está bloqueado en el entorno Diseño. El desarrollador debe explícitamente hacer clic en el icono para desbloquearlo para poder acceder a los objetos del formulario. Esta operación vuelve obsoletos los formularios de usuario relacionados, los cuales deben generarse nuevamente. Cuando un formulario "fuente" se borra, los formularios de usuario relacionados también se borran.*

Los formularios de usuario se almacenan en un archivo independiente con una extensión .4DA, junto al archivo de estructura principal (.4DB/.4DC). Este archivo se llama "archivo de estructura del usuario". El funcionamiento de este archivo es transparente: 4D utiliza un formulario de usuario cuando existe (el nuevo comando ***LIST USER FORMS*** *permite conocer los formularios de usuario válidos en cualquier momento). Es en este archivo también que los comandos* ***FORM SET INPUT*** *y* ***FORM SET OUTPUT*** *buscan los formularios de usuario. Cuando un formulario de usuario es obsoleto, se borra y 4D utiliza el formulario fuente por defecto.*

En cliente/servidor, el archivo .4DA se distribuye en los equipos cliente siguiendo las mismas reglas que el archivo de estructura principal.

Este principio permite conservar los formularios de usuario no obsoletos en caso de una actualización de la estructura por el desarrollador.

Códigos de errores

Los códigos de errores específicos pueden ser devueltos durante la utilización de los comandos de gestión de formularios de usuario. Estos códigos, ubicados en el intervalo de -9750 a -9759, se describen en la sección de ***Errores de la base de datos****.*

Formularios de usuario y formularios de proyecto

Los mecanismos de los formularios de usuario no son compatibles con los formularios de proyecto. Los comandos del tema "Formularios de usuario" no pueden utilizarse con los formularios de proyecto.

CREATE USER FORM

CREATE USER FORM (tabla ; formulario ; formUsuario)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del formulario fuente
formulario	Cadena	➡	Nombre del formulario de tabla fuente
formUsuario	Cadena	➡	Nombre del nuevo formulario usuario

Descripción

*El comando **CREATE USER FORM** duplica el formulario de tabla 4D cuya tabla y nombre se pasan como parámetros y crea un nuevo formulario de usuario llamado formUsuario.*

*Una vez creado, el formulario formUsuario puede modificarse utilizando el comando **EDIT FORM**. Este comando permite crear N formularios de usuarios (por ejemplo, varios formularios de informes) a partir de un mismo formulario fuente.*

Variables y conjuntos del sistema

La variable OK devuelve 1 si la operación se ejecuta correctamente; de lo contrario, devuelve 0.

Gestión de errores

Un error se genera si:

- *formulario ya es un formulario de usuario,*
- *el nombre del formulario de usuario formUsuario es el mismo que el del formulario fuente o que el de un formulario de usuario existente,*
- *el usuario no puede acceder al formulario porque no tiene los derechos de accesos apropiados.*

*Puede interceptar estos errores con el método de gestión de errores instalado por el comando **ON ERR CALL**.*

DELETE USER FORM

DELETE USER FORM (tabla ; formulario ; formUsuario)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del formulario de usuario
formulario	Cadena	➡	Nombre del formulario de tabla fuente
formUsuario	Cadena	➡	Nombre del formulario de usuario

Descripción

*El comando **DELETE USER FORM** permite borrar el formulario de usuario definido por los parámetros tabla, formulario y formUsuario.*

Variables y conjuntos del sistema

Si el formulario de usuario se borra correctamente, la variable OK devuelve 1. De lo contrario, OK toma el valor 0.

Gestión de errores

Se genera un error si:

- *el formulario de usuario no existe o formUsuario contiene una cadena vacía (-9757),*
- *el usuario no tiene los derechos de acceso necesarios para borrar el formulario de usuario.*
  *Puede interceptar este error con el método de gestión de errores instalado por el comando **ON ERR CALL**.*

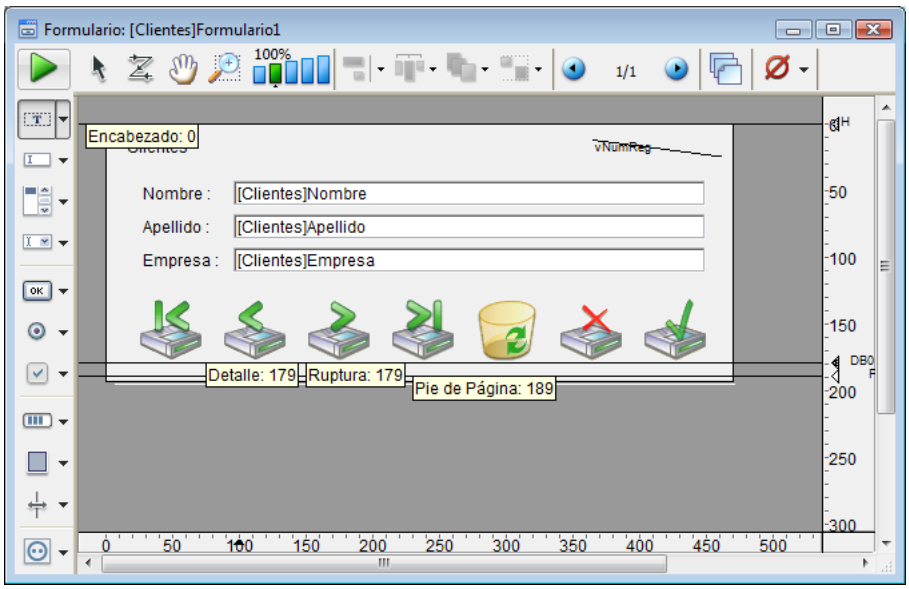
EDIT FORM

EDIT FORM (tabla ; formulario {; formUsuario {; libreria}})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del formulario a modificar
formulario	Cadena	➡	Nombre del formulario de tabla a modificar
formUsuario	Cadena	➡	Nombre del formulario de usuario a modificar
libreria	Cadena	➡	• Ruta de acceso completa de la librería de objetos

Descripción

*El comando **EDIT FORM** abre el formulario de tabla definido por los parámetros tabla, formulario y el parámetro opcional formUsuario en el editor de formularios de usuario :*



***Nota:** la ventana del editor abre únicamente si es la primera ventana del proceso. En otras palabras, generalmente será necesario abrir un nuevo proceso para mostrar el editor.*

Si pasa una cadena vacía en el parámetro formUsuario y si no hay un formulario de usuario relacionado con formulario, el formulario fuente se muestra en el editor. El formulario modificado se copia entonces en el archivo de estructura de usuario (.4DA) y será utilizado como un remplazo de formulario.

*Si un formulario de usuario ya ha sido generado a partir de formulario utilizando este comando, el formulario usuario se muestra en el editor. Si quiere comenzar desde el formulario fuente, primero debe eliminar el formulario de usuario utilizando el comando **DELETE USER FORM**.*

*El parámetro formUsuario permite definir un formulario de usuario (creado utilizando el comando **CREATE USER FORM**) a modificar. En este caso, el formulario se muestra en el editor.*

En el parámetro opcional libreria, pase la ruta de acceso completa de la librería de objetos que el usuario estará autorizado a utilizar para personalizar el formulario. Cuando se utilizan con el editor de formularios de usuario, las librerías de objetos permiten pegar objetos con sus propiedades gráficas y sus acciones automáticas. Los objetos con métodos no aparecen en la librería. Atención, el desarrollador debe verificar que la adición de los objetos de una librería sea compatible con el formulario de usuario (y sus objetos) a nivel de nombres, variables y tipos.

*En modo cliente/servidor, la librería debe encontrarse en la carpeta **Extras** de la base, en el mismo nivel de la carpeta **Plugins**, de manera que esté disponible para todos los equipos clientes. Si la librería es válida, se abre desde la ventana del formulario.*

Para mayor información sobre la librería de imágenes, consulte el Manual de Diseño.

Ejemplo

En este ejemplo, el usuario puede elegir una librería y luego modificar un formulario de diálogo:

```
MAP FILE TYPES("4DLB";"4IL";"Libreria 4D")
$vALib:=Select document(1;"4DLB";"Por favor seleccione una librería";0)
If(OK=1)
  `Se eligió una librería
   $vARutaLib:=Document
Else
   $vARutaLib:=""
End if

EDIT FORM([Dialogos];"Bienvenido";"Lib_Logos.4il")
If(OK=1)
  `Presentación del formulario modificado
```

```
    DIALOG([Dialogos];"Bienvenido")
End if
```

Variables y conjuntos del sistema

Si el usuario guarda las modificaciones al formulario, la variable OK toma el valor 1. En caso de error, OK toma el valor 0.

Gestión de errores

Se genera un error si:

- *el formulario no ha sido declarado como modificable por el usuario en el entorno Diseño o si no existe,*
- *el formulario está abierto y está siendo modificado por otro proceso,*
- *el usuario no puede acceder al formulario porque no tiene los derechos de acceso necesarios.*

*Puede interceptar este error con el método de gestión de errores instalado por el comando **ON ERR CALL**.*

LIST USER FORMS

LIST USER FORMS (tabla ; formulario ; arrayFormUsuarios)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del formulario fuente
formulario	Cadena	➡	Nombre del formulario de tabla fuente
arrayFormUsuarios	Array cadena	⬅	Nombres de los formularios de usuario que provienen del formulario fuente

Descripción

*El comando **LIST USER FORMS** llena el array arrayFormUsuarios con los nombres de los formularios de usuario que provienen del formulario del desarrollador (formulario de tabla) definido por los parámetros tabla y formulario.*

*Si el formulario de usuario fue creado directamente utilizando el comando **EDIT FORM**, el único elemento que contiene arrayFormUsuarios es una cadena vacía ("").*

El array se devuelve vacío si no hay ningún formulario de usuario para el formulario del desarrollador especificado.

Fórmulas

- *Utilizar tokens en fórmulas*
- *EDIT FORMULA*
- *EXECUTE FORMULA*
- *GET ALLOWED METHODS*
- *Parse formula* *New 17.0*
- *SET ALLOWED METHODS*

Utilizar tokens en fórmulas

Presentación

El lenguaje de 4D incluye un sistema de tokenización único para todos los nombres de los objetos del lenguaje que se utilizan en el código (constantes, comandos, tablas, campos y palabras claves). Tokenizar estos nombres significa almacenarlos internamente como referencias absolutas (números) y luego se restauran durante la ejecución o visualización en función del contexto. Esto le permite garantizar que el código siempre será interpretado correctamente, incluso si cambia el nombre de las tablas o campos, o cuando los comandos del lenguaje 4D cambian de nombre a lo largo de las diferentes versiones de la aplicación.

Nota: *esto también garantiza la traducción automática del código cuando se ha activado la opción "Usar configuración del sistema regional" en la* ***Página Métodos*** *de las Preferencias y abrir sus bases con las versiones de 4D en diferentes idiomas.*

La Tokenización es completamente transparente para los desarrolladores 4D al trabajar en el editor de código. Sin embargo, este mecanismo no se aplica de forma automática en las fórmulas 4D ya que consisten en texto que se interpreta durante la ejecución y no cuando se escribe. De hecho, este es el caso tan pronto como el código 4D se expresa en forma de texto sin formato, más específicamente cuando se exporta código y luego se importa utilizando los comandos ***METHOD GET CODE*** *y* ***METHOD SET CODE****, copiar/pegar o interpretado desde* ***Etiquetas HTML 4D****.*
Para seguir beneficiándose de los mecanismos de tokenización en estos contextos, sólo tiene que utilizar una sintaxis explícita (descrita más adelante), que consiste en preceder los nombres de los objetos del lenguaje de su token.

Sintaxis tokenizada

Por defecto, el mecanismo de tokens no se implementa automáticamente en las fórmulas 4D (así como en los contextos donde el código 4D se expresa en forma de texto sin formato, ver abajo). Por consiguiente, 4D propone, para los elementos llamados contenidos en las expresiones, una sintaxis especial que puede utilizar para referenciar directamente los tokens: sólo tiene que añadir un sufijo específico después el nombre del elemento para indicar su tipo (comando, campo, etc.), seguido por su referencia. La ***sintaxis tokenizada*** *se detalla en la siguiente tabla:*

Elemento	Ejemplo (sintaxis estándar)	Sufijo	Ejemplo (sintaxis tokenizada)	Comentarios
Comando 4D	Cadena	:Cxx	String:C10(a)	xx es el número del comando
Tabla	[Employees]	:xx	[Employees:1]	xx es el número de la tabla
Campo	[Employees]Name	:xx	[Employees:1]Name:2	xx es el número del campo
Plugin 4D	PV PRINT(area)	:Pxx:yy	PV PRINT:P13000:229(area)	xx es el ID del plug-in y yy es el índice del comando

Nota: *las letras mayúsculas (C, P) deben utilizarse en los sufijos; de lo contrario, no se interpretarán correctamente.*

Cuando se utiliza esta sintaxis, usted garantiza que sus fórmulas se interpretarán correctamente incluso en caso de cambio de nombre o cuando la base de datos se ejecuten en un lenguaje diferente.

Nota: *las constantes también se tokenizan en el lenguaje sin embargo, en las fórmulas sólo puede pasar su valor con el fin de hacerlas independientes del contexto.*

Esta sintaxis es aceptada en todas las fórmulas 4D (o expresiones 4D), independientemente del contexto de llamada:

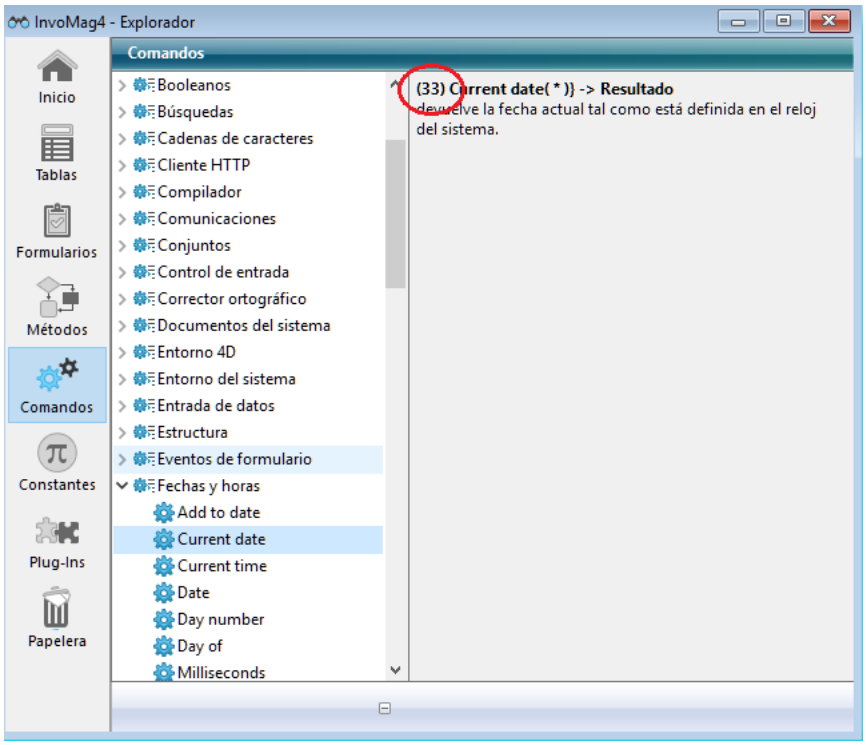
- *fórmulas 4D ejecutadas utilizando el* ***Editor de fórmulas*** *o utilizando los comandos tales como* ***EXECUTE FORMULA****,* ***APPLY TO SELECTION****,* ***QUERY BY FORMULA****,* ***LISTBOX INSERT COLUMN FORMULA****, etc.*
- *expresiones insertadas en áreas de texto enriquecido (ver* ***ST INSERT EXPRESSION*** *y* ***Etiquetas soportadas****),*
- *expresiones calculadas en las etiquetas de transformación (ver* ***Etiquetas HTML 4D****),*
- *expresiones insertadas en áreas de plug-ins,*
- *expresiones insertadas en áreas 4D Write Pro (a partir de 4D v15).*

¿Dónde encontrar los números de los elementos?

La sintaxis tokenizada requiere la adición de los números de referencia de los elementos. La ubicación de estas referencias depende del tipo del elemento.

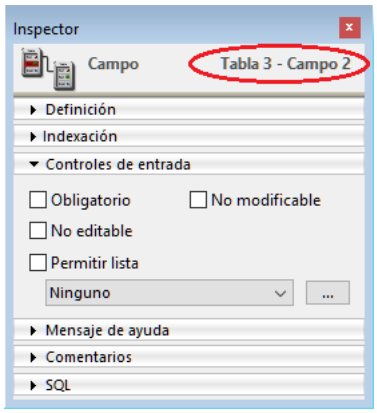
Comandos 4D

Los números de los comandos se pueden encontrar en este manual Lenguaje (área "Propiedades"), así como en la página ***Comandos*** *del Explorador:*



Tablas y campos

*Los números de tablas y de campos se pueden obtener utilizando los comandos **Table** y **Field**.*
*También se muestran en la **Inspector** del editor de estructura:*



Comandos de plug-ins 4D

Para saber cuáles son los tokens de los comandos de plug-ins 4D, el truco consiste en introducir el código deseado en el editor de métodos y reiniciar 4D después de desactivar el plug-in (por ejemplo, moviendo su carpeta). Esto significa que sólo los tokens se mostrarán en el editor de métodos, y a continuación, puede copiar los que necesite.

Código con plug-in instalado:

```
12      PV SET ROWS HEIGHT (Area;1;10;PV Get row height (Area;10)+$Height)
```

El mismo código con inactivación del plug-in:

```
12      Ô13000;75Ô (Area;1;10;Ô13000;76Ô (Area;10)+$Height)
```

EDIT FORMULA

EDIT FORMULA (tabla ; formula)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a mostrar por defecto en el editor de fórmulas
formula	Texto	➡	Variable que contiene la fórmula a mostrar en el editor de fórmulas o "" para mostrar sólo el editor
		⬅	Fórmula validada por el usuario

Descripción

El comando EDIT FORMULA muestra el editor de fórmulas con el fin de permitir al usuario escribir o modificar una fórmula. El editor contiene al abrir:

- *en la lista de la izquierda, los campos de la tabla pasados en el parámetro tabla,*
- *en el área de fórmula, la fórmula contenida en la variable formula. Si pasa una cadena vacía en formula, el editor es mostrado sin fórmula.*

El usuario puede modificar y guardar la formula mostrada. También es posible escribir o cargar una nueva fórmula. Sin embargo, si el usuario valida la caja de diálogo, la variable de sistema OK toma el valor 1 y la variable formula contiene la fórmula definida por el usuario. Si el usuario cancela la fórmula, la variable sistema OK toma el valor 0 y la variable formula no cambia.

Notas:

- *Por defecto, el acceso a los métodos y a los comandos está restringido para todos los usuarios (excepto para el Diseñador y Administrador, en bases de datos creadas con 4D 2004.4 y posteriores). Cuando este mecanismo está activado, usted debe designar explícitamente los elementos accesibles a los usuarios utilizando el comando **SET ALLOWED METHODS**. Si la formula llama métodos que no han sido autorizados previamente en el editor de fórmulas utilizando el comando **SET ALLOWED METHODS**, se genera un error de sintaxis y no podrá validar la caja de diálogo.*
- *El editor de fórmulas no está asociado a ninguna barra de menús de forma predeterminada. Debe instalar un menú **Editar** estándar en el proceso de llamada si desea que los usuarios puedan beneficiarse de los accesos directos cortar/copiar/pegar en el editor de fórmulas.*

*Recuerde que cuando se valida la caja de diálogo, el comando no ejecuta la formula; sólo valida y actualiza el contenido de la variable. Si quiere ejecutar la formula, debe utilizar el comando **EXECUTE FORMULA**.*

Ejemplo

Visualización del editor de fórmulas con la tabla [Empleados] y sin una fórmula introducida previamente:

```
$miFormula:=""
EDIT FORMULA([Empleados];$miFormula)
If(OK=1)
   APPLY TO SELECTION([Empleados];EXECUTE FORMULA($miFormula))
End if
```

Variables y conjuntos del sistema

Si el usuario valida la caja de diálogo, la variable sistema OK toma el valor 1. Si el usuario anula la caja de diálogo, la variable sistema OK toma el valor 0.

EXECUTE FORMULA

EXECUTE FORMULA (instruccion)

Parámetro	Tipo		Descripción
instruccion	Cadena	➡	Código a ejecutar

Descripción

EXECUTE FORMULA ejecuta instruccion como una línea de código. Este comando está diseñado para ser utilizado cuando necesita evaluar expresiones que el usuario puede construir o modificar.

La cadena de instrucción debe ser de una sola línea. Si instruccion es una cadena vacía, **EXECUTE FORMULA** no hace nada. La regla es que si la instrucción puede ejecutarse como un método de una línea, entonces se ejecutará correctamente. El comando **EXECUTE FORMULA** debe utilizarse con precaución, ya que disminuye la velocidad de ejecución. En una base compilada, el código de la instrucción no está compilado. Esto significa que la instrucción será ejecutada, pero no será verificada por el compilador en el momento de la compilación.

Nota: la ejecución de fórmulas en modo compilado se puede optimizar utilizando una memoria caché (ver **Caché para fórmulas en modo compilado** abajo).

La instrucción puede incluir los siguientes elementos:

- una llamada a una función (método proyecto que devuelve un valor),
- una llamada a un comando 4D
- una tarea

La fórmula puede incluir variables proceso e interproceso. La instrucción no puede contener instrucciones de control de flujo (If, While, etc.), porque la instrucción debe tener sólo una línea de código.
Para asegurarse de que la instruccion sea evaluada correctamente independientemente del lenguaje o versión 4D utilizada, se recomienda utilizar la sintaxis tokenizada para los elementos cuyo nombre puede variar entre las diferentes versiones (comandos, tablas, campos, constantes). Por ejemplo, para insertar el comando **Current time**, introduzca **'Current time:C178'**. Para más información, consulte **Utilizar tokens en fórmulas**.

Notas:

- Si la instruccion es un método proyecto, se recomienda utilizar **EXECUTE METHOD** que le permite pasar parámetros.
- No se recomienda llamar a ningún comando de declaración de variable como **C_DATE** en instruccion ya que puede generar conflictos en el código.

La fórmula puede incluir variables de proceso y variables entre procesos. Sin embargo, la declaración no puede contener el control de las instrucciones de flujo (If, While, etc.), ya que debe estar en una línea de código.

Para garantizar que la instrucción se evalúe correctamente, independientemente del lenguaje 4D o la versión utilizada, se recomienda utilizar la sintaxis del token para los elementos cuyo nombre puede variar entre diferentes versiones (comandos, tablas, campos, constantes). Por ejemplo, para insertar el comando [#cmd id="178"/], introduzca '**Current time:C178**'. Para más información al respecto, consulte **Utilizar tokens en fórmulas**.

Caché para fórmulas en modo compilado

Por optimización, cada fórmula ejecutada vía **EXECUTE FORMULA** en modo compilado se puede almacenar en una memoria caché en la memoria dedicada. La fórmula se almacena en caché en forma tokenizada. Una vez que se coloca en la caché, sus ejecuciones posteriores están altamente optimizadas ya que el paso de tokenización se evita.

El tamaño de la caché es cero por defecto (sin caché); debe ser creada o ajustada con el comando **SET DATABASE PARAMETER**. Por ejemplo:

```
SET DATABASE PARAMETER(Number of formulas in cache;0) //sin caché de fórmulas
SET DATABASE PARAMETER(Number of formulas in cache;3) //hasta tres fórmulas se puede almacenar en caché para cada proceso
```

El comando **EXECUTE FORMULA** utiliza la caché sólo cuando se llama desde una base o componente compilado.

Ejemplo

Usted desea ejecutar una fórmula incluida las llamadas a los comandos y tablas 4D. Dado que estos elementos potencialmente podrían ser renombrados, quiere asegurarse de la correcta ejecución de la instrucción en las versiones futuras de su aplicación utilizando la sintaxis tokens:

```
EXECUTE FORMULA("Year of:C25 ([Products:5]Creation_Date:2])+$add")
```

GET ALLOWED METHODS

GET ALLOWED METHODS (arrMetodos)

Parámetro	Tipo		Descripción
arrMetodos	Array cadena	⇐	Array de nombres de métodos

Descripción

*El comando **GET ALLOWED METHODS** devuelve, en arrMetodos, los nombres de los métodos que pueden utilizarse para escribir fórmulas. Estos métodos están listados al final de la lista de comandos en el editor.*

*Por defecto, los métodos no pueden ser utilizados en el editor de fórmulas. Los métodos deben ser autorizados explícitamente utilizando el comando **SET ALLOWED METHODS**. Si este comando no ha sido ejecutado, **GET ALLOWED METHODS** devuelve un array vacío.*

***GET ALLOWED METHODS** devuelve exactamente lo que se le pasó a **SET ALLOWED METHODS**, es decir un array alfa (el comando crea y dimensiona el array). Igualmente, si el carácter arroba (@) se utiliza para definir un grupo de métodos, se devuelve la cadena que contiene el carácter @ (y no los nombres de los métodos del grupo).*

Este comando es útil para conservar los parámetros del conjunto actual de métodos autorizados antes de la ejecución de una fórmula en un contexto específico (por ejemplo, un informe rápido).

Ejemplo

Este ejemplo autoriza un conjunto de métodos específicos para crear un informe:

```
 `Almacenamiento de los parámetros actuales
GET ALLOWED METHODS(metodosArray)

 `Definición de los métodos para el informe rápido
arrMetodos_Reports{1}:="Reports_@"
SET ALLOWED METHODS(arrMetodos_Reports)
QR REPORT([Personas];"MiInforme")

 `Reestablecimiento de los parámetros actuales
SET ALLOWED METHODS(arrMetodos)
```

Parse formula

Parse formula (formula {; opciones}{; mensajeError}) -> Resultado

Parámetro	Tipo		Descripción
formula	Texto	➡	Fórmula de texto sin formato
opciones	Entero largo	➡	Instrucciones para entrada/salida
mensajeError	Texto	⬅	Mensaje de error (cadena vacía si no hay error)
Resultado	Texto	↩	Fórmula transformada (texto plano)

Descripción

*La función **Parse formula** analiza la formula 4D, verifica su sintaxis y devuelve su formulario normalizado. Esto permite que la fórmula siga siendo válida en el caso de que se cambie el nombre de un lenguaje 4D o elemento de estructura (comando, constante, tabla, campo o Plugin 4D).*

***Parse formula** se puede usar para evaluar y traducir fórmulas de la siguiente manera:*

- *Los nombres reales de tabla/campo se pueden convertir a nombres de estructura* virtual (nombres personalizados) o equivalentes tokenizados ***
- *Los equivalentes de tabla/campo tokenizados se pueden convertir a nombres de estructura virtual o nombres reales de tabla/campo*
- *Las estructuras virtuales se pueden convertir a nombres reales de tabla/campo o equivalentes simbólicos*
- *Los elementos de lenguaje 4D se pueden convertir a equivalentes de lenguaje 4D tokenizados*
- *Los equivalentes del lenguaje 4D tokenizados se pueden convertir a elementos de lenguaje 4D*

** Las estructuras virtuales se definen utilizando los comandos **SET TABLE TITLES** y **SET FIELD TITLES** (* parámetro requerido).*

*** Los equivalentes tokenizados son elementos de estructura y del lenguaje 4D en texto sin formato expresados con la sintaxis del token, como se muestra a continuación (consulte también **Utilizar tokens en fórmulas**):*

```
[Table:1]Field:1+String:C10(1)
```

En formula, pase una fórmula en texto sin formato. Puede usar nombres de estructuras reales o virtuales, así como equivalentes tokenizados.

*Sin importar los tipos de nombre utilizados en formula, por defecto **Parse formula** devuelve el lenguaje 4D real o los nombres de los elementos de la estructura sin tokens de texto.*

*El parámetro opcional opciones le permite especificar cómo se expresa y/o se devuelve formula utilizando las siguientes constantes del tema **Fórmulas**. Puede combinar constantes para designar tanto el formato de entrada como el de salida de la fórmula devuelta.*

Constante	Valor	Comentario
Formula in with virtual structure	1	La fórmula contiene nombres personalizados (virtual). Por defecto, la fórmula devuelta contiene nombres reales.
Formula out with virtual structure	2	La fórmula devuelta debe contener nombres personalizados (virtual).
Formula out with tokens	4	La fórmula devuelta debe contener texto tokenizado (por ejemplo: Cxx).

El parámetro opcional mensajeError recibirá un mensaje de error si hay un error de sintaxis en formula. Si no hay ningún error, se devolverá una cadena vacía.

Ejemplo 1

```
ARRAY TEXT($t1;1)
ARRAY LONGINT($t2;1)
$t1{1}:="Virtual table"
$t2{1}:=1
SET TABLE TITLES($t1;$t2;*)

ARRAY TEXT($tf1;1)
ARRAY LONGINT($tf2;1)
$tf1{1}:="Virtual field"
$tf2{1}:=2
SET FIELD TITLES([Table_1];$tf1;$tf2;*)

 //Estructura virtual para tabla y nombre de campo equivalente
$parsedFormula:=Parse formula("[Virtual table]Virtual field";Formula in with virtual structure;$errorMessage)
 //return [Table_1]Field_2

 //Nombre de tabla y campo para equivalente de estructura virtual
$parsedFormula:=Parse formula("[Table_1]Field_2";Formula out with virtual structure;$errorMessage)
 //return [Virtual table]Virtual field
```

```
 //Tabla y nombre de campo para el formulario tokenizado equivalente
$parsedFormula:=Parse formula("String([Table_1]Field_2)";Formula out with tokens;$errorMessage)
 //return String:C10([Table_1:1]Field_2:2)
```

Ejemplo 2

*Utilizando las tablas del **Ejemplo 1**:*

```
 //pide al usuario que escriba su fórmula favorita
$formula:=""
EDIT FORMULA([Table_1];$formula)

 //guarda la fórmula del usuario para un uso posterior
CREATE RECORD([users_preferences])
$persistentFormula:=Parse formula($formula;Formula out with tokens)
[users_preferences]formula:=$persistentFormula

 //luego: ejecuta la fórmula guardada anteriormente
CREATE RECORD([Table_1])
EXECUTE FORMULA([users_preferences]formula)
```

SET ALLOWED METHODS

SET ALLOWED METHODS (arrMetodos)

Parámetro	Tipo		Descripción
arrMetodos	Array texto	➡	Array de nombres de métodos

Descripción

*El comando **SET ALLOWED METHODS** permite designar los métodos proyecto que pueden ser llamados directamente desde la aplicación.*

4D incluye un mecanismo de seguridad que filtra los métodos proyecto llamables de los siguientes contextos:

- *El editor de fórmulas: los métodos permitidos aparecen al final de la lista de comandos por defecto y pueden utilizarse en fórmulas (ver la sección **Descripción del editor de fórmulas**).*
- *Documentos 4D Write Pro: los métodos permitidos se pueden utilizar en las expresiones dinámicas insertadas en los documentos (ver la sección **Filtrar expresiones contenidas en un documento 4D Write Pro**).*
- *El editor de etiquetas 64 bits: los métodos permitidos aparecen en el menú **Aplicar** si también se comparten con el componente (ver la sección **Descripción del editor de etiquetas**).*

*Por defecto, si no utiliza el comando **SET ALLOWED METHODS**, ningún método es llamable (utilizar un método no autorizado en una expresión, genera un error de sintaxis).*

En el Parámetro arrMetodos pase el nombre del array que contiene la lista de métodos a ofrecer en el editor de fórmulas. El array debe haberse definido previamente.

Puede utilizar el carácter arroba (@) en los nombres de los métodos para definir uno o más grupos de métodos autorizados.

Si quiere que el usuario pueda llamar los comandos 4D no autorizados por defecto o a los comandos de plug-ins, debe utilizar los métodos específicos encargados de ejecutar estos comandos.

***Nota**: el mecanismo de restricción de acceso a los comandos y métodos en el editor de formularios puede ser desactivado por todos los usuarios o por el Diseñador y Administrador a través de una opción en la página "Seguridad" de las Propiedades de la base. Si la opción "Desactivada para todos" está seleccionada, el comando **SET ALLOWED METHODS** no tendrá efecto.*

Ejemplo

Este ejemplo autoriza todos los métodos que comienzan por “formula” y el método “Total_general”:

```
ARRAY TEXT(arrMetodos;2)
arrMetodos{1}:="formula@"
arrMetodos{2}:="Total_general"
SET ALLOWED METHODS(arrMetodos)
```

Funciones estadísticas

- *On a Series*
- *Average*
- *Max*
- *Min*
- *Std deviation*
- *Sum*
- *Sum squares*
- *Variance*

On a Series

Las funciones de este tema realizan cálculos sobre una serie de valores.

*Las funciones **Average**, **Min**, **Max**, **Sum**, **Sum squares**, **Std deviation** y **Variance** se aplican a campos o arrays:*

- *cuando se aplican a los campos, utilizan la selección actual de registros,*
- *cuando se aplican a arrays, utilizan los elementos del array.*

*Note que cuando se aplican a campos, las funciones **Sum squares**, **Std deviation** y **Variance** sólo pueden ser utilizadas durante la impresión.*

Estas funciones trabajan sólo con valores numéricos y devuelve un valor numérico.

Uso de funciones estadísticas aparte de la impresión

*Cuando las funciones **Average**, **Min**, **Max** y **Sum** se utilizan en un campo fuera de una operación de impresión, puede que tengan que cargar cada registro en la selección actual para calcular el resultado. Si hay muchos registros, este proceso puede tomar mucho tiempo. Para limitar el tiempo del proceso, puede indexar el campo.*

***Nota**: cuando la operación es larga, aparece un termómetro de progreso. Este termómetro tiene un botón Detener que permite al usuario interrumpir la operación. Si el usuario hace clic en este botón, la variable OK toma el valor 0. Si la operación se completa correctamente, la variable OK toma el valor 1.*

Uso de funciones estadísticas en un informe impreso

*Cuando se utilizan funciones estadísticas en un informe, se comportan de forma diferente porque el informe mismo debe cargar cada registro. Utilice estas funciones en un formulario o método de objeto cuando imprima con el comando **PRINT SELECTION** o cuando imprima seleccionando **Imprimir** en el menú **Archivo** en el entorno Diseño.*

Cuando utilice estas funciones en un informe, los valores devueltos sólo son confiables en el nivel de ruptura 0 y cuando el proceso de rupturas esté activo. Esto significa que sólo son útiles al final del informe, cuando todos los registros hayan sido procesados.

Utilice estas funciones sólo en un método de objeto para un área no editable incluida en el área de ruptura B0.

Recuerde que un campo pasado como parámetro a una función estadística debe ser numérico.

Average

Average (series {; rutaAtributo}) -> Resultado

Parámetro	Tipo		Descripción
series	Campo, Array	➡	Datos para los cuales se devuelve el promedio
rutaAtributo	Texto	➡	Ruta del atributo del cual calcular el promedio
Resultado	Real	⮌	Media aritmética (promedio) de series

Descripción

Average devuelve la media aritmética (promedio) de series. Si series es un campo indexado, el índice se utiliza para calcular el promedio.

Puede pasar en series un array (de una o dos dimensiones). En este caso, el array debe ser de tipo Entero, Entero largo o Real.

Este comando acepta un parámetro opcional de tipo texto rutaAtributo, que puede utilizar si series es un campo de tipo Objeto. Le permite definir la ruta del atributo a calcular. Utilice la notación estándar para definir las rutas de los atributos anidados, por ejemplo "company.address.number". Recuerde que los nombres de los atributos de objetos tienen en cuanta las mayúsculas y minúsculas.
Sólo los valores numéricos de los atributos se utilizan para el calculo. Si hay valores en la ruta del atributo que no son de tipo numérico, se omiten.

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. Si se interrumpe (por ejemplo si el usuario hace clic en el botón **Detener** del termómetro de progreso) la variable OK toma el valor 0.

Ejemplo 1

El siguiente ejemplo define la variable vPromedio que está en el área de ruptura B0 de un formulario de salida. La línea de código es el método de objeto para vPromedio. El método de objeto no se ejecuta hasta el nivel de ruptura 0:

```
vPromedio:=Average([Empleados] Salario)
```

El siguiente método se llama para imprimir los registros en la selección y activar el proceso de ruptura:

```
ALL RECORDS([Empleados])
ORDER BY([Empleados];[Empleados]Apellido;>)
BREAK LEVEL(1)
ACCUMULATE([Empleados]Salario)
FORM SET OUTPUT([Empleados];"Imprimir formulario")
PRINT SELECTION([Empleados])
```

Nota: el parámetro del comando **BREAK LEVEL** debe ser igual al número de rupturas en su informe. Para mayor información sobre rupturas, consulte **Impresión**.

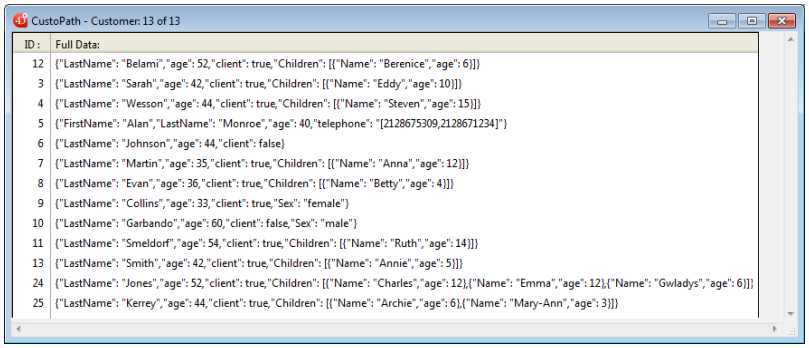
Ejemplo 2

Este ejemplo permite obtener la media de los 15 primeras notas de la selección:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
ORDER BY([Exams];[Exams]Exam_Grade;<)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
ARRAY REAL($ArrGrades;15)
vAverage:=Average($ArrGrades)
```

Ejemplo 3

Su tabla [Customer] contiene un campo objeto "full_Data" con los siguientes datos:



Puede hacer los siguientes cálculos:

```
C_REAL($vAvg)
ALL RECORDS([Customer])
$vAvg:=Average([Customer]full_Data;"age")
 //$vAvg is 44,46

C_LONGINT($vTot)
$vTot:=Sum([Customer]full_Data;"Children[].age")
 //$vTot is 105
```

Max

Max (series {; rutaAtributo}) -> Resultado

Parámetro	Tipo		Descripción
series	Campo, Array	➡	Datos para los cuales se devuelve el valor máximo
rutaAtributo	Texto	➡	Ruta de atributo para el cual calcular el valor máximo
Resultado	Fecha, Número	⮌	Máximo valor en series

Descripción

Max *devuelve el valor máximo en series. Si series es un campo indexado, el índice se utiliza para buscar el valor máximo.*

Puede pasar un array (una o dos dimensiones) en series. En este caso, el array debe ser del tipo Entero, Entero largo, Real o Fecha.

Si la selección de series está vacía, ***Max*** *devuelve 0.*

Este comando acepta un parámetro opcional de tipo texto rutaAtributo, que puede utilizar si series es un campo de tipo Objeto. Le permite definir la ruta del atributo a calcular. Utilice la notación estándar para definir las rutas de los atributos anidados, por ejemplo "company.address.number". Recuerde que los nombres de los atributos de objetos tienen en cuanta las mayúsculas y minúsculas.
Sólo los valores numéricos de los atributos se utilizan para el calculo. Si hay valores en la ruta del atributo que no son de tipo numérico, se omiten.

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. Si se interrumpe (por ejemplo si el usuario hace clic en el botón ***Detener*** *del termómetro de progreso), la variable OK toma el valor 0.*

Ejemplo 1

El siguiente ejemplo es un método de objeto de la variable vMax ubicada en el área de ruptura 0 del formulario. La variable se imprime al final del informe. El método de objeto asigna el valor máximo de campo a la variable, el cual se imprime en la última ruptura del informe.

```
vMax:=Max([Employees] Salary)
```

Nota: *asegúrese de que el evento formulario "On printing break" esté seleccionado para la variable.*
El siguiente método se llama para imprimir los registros de la selección y activar el proceso de ruptura:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]Company;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
FORM SET OUTPUT([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

Nota: *el parámetro del comando* ***BREAK LEVEL*** *debe ser igual al número de rupturas en su informe. Para mayor información sobre rupturas, consulte el capítulo* ***Impresión****.*

Ejemplo 2

Este ejemplo permite obtener el valor más elevado de un array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vMax:=Max($ArrGrades)
```

Ejemplo 3

Para un ejemplo de cálculo de un atributo campo de objeto, consulte el ejemplo 3 del comando ***Average****.*

Min

Min (series {; rutaAtributo}) -> Resultado

Parámetro	Tipo		Descripción
series	Campo, Array	➡	Datos para los cuales devuelve el valor mínimo
rutaAtributo	Texto	➡	Ruta de atributo para el cual calcular el valor mínimo
Resultado	Fecha, Número	⮌	Valor mínimo en series

Descripción

Min devuelve el valor mínimo en series. Si series es un campo indexado, el índice se utiliza para encontrar el valor mínimo.

Si la selección de series está vacía, **Min** devuelve 0.

Puede pasar en series un array (de una o dos dimensiones). En este caso, el array debe ser de tipo Entero, Entero largo, Real o Fecha.

Este comando acepta un parámetro opcional de tipo texto rutaAtributo, que puede utilizar si series es un campo de tipo Objeto. Le permite definir la ruta del atributo a calcular. Utilice la notación estándar para definir las rutas de los atributos anidados, por ejemplo "company.address.number". Recuerde que los nombres de los atributos de objetos tienen en cuanta las mayúsculas y minúsculas.
Sólo los valores numéricos de los atributos se utilizan para el calculo. Si hay valores en la ruta del atributo que no son de tipo numérico, se omiten.

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. Si se interrumpe (por ejemplo si el usuario hace clic en el botón **Detener** del termómetro de progreso), la variable OK toma el valor 0.

Ejemplo 1

El siguiente ejemplo es un método de objeto para la variable vMin ubicada en el área de ruptura 0 del formulario. La variable se imprime al final del informe. El método de objeto asigna el valor mínimo del campo a la variable, el cual se imprime en la última ruptura del informe:

```
vMin:=Min([Employees]Salary)
```

Note: asegúrese de que el evento formulario "On printing break" esté seleccionado para la variable.

El siguiente método se llama para imprimir los registros en la selección y activar el proceso de ruptura:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]Company;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
FORM SET OUTPUT([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

Nota: el parámetro del comando **BREAK LEVEL** debe ser igual al número de rupturas en su informe. Para mayor información sobre rupturas, consulte el capítulo **Impresión**.

Ejemplo 2

El siguiente ejemplo busca la venta más baja de un empleado y muestra el resultado en una caja de diálogo de alerta. Las cantidades vendidas son guardadas en el subcampo [Empleados]VentasDolares:

```
ALERT("Ventaminima = "+String(Min([Empleados]VentasDolares)))
```

Ejemplo 3

Este ejemplo obtiene el valor mínimo en el array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vMin:=Min($ArrGrades)
```

Ejemplo 4

Para un ejemplo de cálculo de un atributo campo de objeto, consulte el ejemplo 3 del comando **Average**.

Std deviation

Std deviation (series) -> Resultado

Parámetro	Tipo		Descripción
series	Campo, Array	➡	Datos para los cuales se devuelve la desviación estándar
Resultado	Real	⮌	Desviación estándar de series

Descripción

Std deviation devuelve la desviación estándar de series.

Si series es un campo indexado, el índice se utiliza para calcular la desviación estándar.
Puede pasar en series un array (de una o dos dimensiones). En este caso, el array debe ser de tipo Entero, Entero largo o Real.

Ejemplo 1

El siguiente ejemplo es un método de objeto para la variable vDesv. El método de objeto asigna la desviación estándar de una serie de datos a vDesv:

```
vDesv:=Std deviation([Tabla1]SeriesDatos)
```

El siguiente método se llama para imprimir los registros en la selección y activar el proceso de ruptura:

```
ALL RECORDS([Tabla1])
ORDER BY([Tabla1];[Tabla1]SeriesDatos;>)
BREAK LEVEL(1)
ACCUMULATE([Tabla1]SeriesDatos)
OUTPUT FORM([Tabla1];"Imprimir formulario")
PRINT SELECTION([Tabla1])
```

Nota: el parámetro del comando **BREAK LEVEL** debe ser igual al número de rupturas de su informe. Para mayor información sobre rupturas, consulte los comandos de impresión.

Ejemplo 2

Este ejemplo obtiene la desviación estándar de una serie de valores ubicados en un array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vStdDev:=Std deviation($ArrGrades)
```

Sum

Sum (series {; rutaAtributo}) -> Resultado

Parámetro	Tipo		Descripción
series	Campo, Array	➡	Datos para los cuales se devuelve la suma
rutaAtributo	Texto	➡	Ruta de atributo para el cual calcular la suma
Resultado	Real	⮌	Suma de series

Descripción

Sum *devuelve la suma (es decir, el total de todos los valores) de series. Si series es un campo indexado, el índice se utiliza para calcular el calculo.*

Puede pasar en series un array (a una o dos dimensiones). En este caso, el array debe ser de tipo Entero, Entero largo o Real.

Este comando acepta un parámetro opcional de tipo texto rutaAtributo, que puede utilizar si series es un campo de tipo Objeto. Le permite definir la ruta del atributo a calcular. Utilice la notación estándar para definir las rutas de los atributos anidados, por ejemplo "company.address.number". Recuerde que los nombres de los atributos de objetos tienen en cuanta las mayúsculas y minúsculas.
Sólo los valores numéricos de los atributos se utilizan para el calculo. Si hay valores en la ruta del atributo que no son de tipo numérico, se omiten.

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. Si se interrumpe (por ejemplo si el usuario hace clic en el botón ***Detener*** *del termómetro de progreso) la variable OK toma el valor 0.*

Ejemplo 1

El siguiente ejemplo es un método de objeto para una variable vTotal ubicada en un formulario. El método de objeto asigna la suma de todos los salarios a vTotal:

```
vTotal:=Sum([Empleados]Salario)
```

El siguiente método se llama para imprimir los registros en la selección y para activar el proceso de ruptura:

```
ALL RECORDS([Empleados])
ORDER BY([Empleados];[Empleados]Apellido;>)
BREAK LEVEL(1)
ACCUMULATE([Empleados]Salario)
OUTPUT FORM([Empleados];"Imprimir formulario")
PRINT SELECTION([Empleados])
```

Nota: *el parámetro para el comando* ***BREAK LEVEL*** *debe ser igual al número de rupturas en su informe. Para mayor información sobre el proceso de rupturas consulte los comandos de impresión.*

Ejemplo 2

Este ejemplo permite obtener la suma de todos los valores ubicados en un array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vSum:=Sum($ArrGrades)
```

Ejemplo 3

Para un ejemplo de cálculo de un atributo campo de objeto, consulte el ejemplo 3 del comando ***Average****.*

Sum squares

Sum squares (series) -> Resultado

Parámetro	Tipo		Descripción
series	Campo, Array	➡	Datos para los cuales se devuelve la suma de cuadrados
Resultado	Real	⮌	Suma de cuadrados de series

Descripción

Sum squares *devuelve la suma de cuadrados de series. Si series es un campo indexado, el índice se utiliza para calcular la suma de cuadrados.*
Puede pasar un array (de una o dos dimensiones) en series. En este caso, el array debe ser de tipo Entero, Entero largo o Real.

Ejemplo 1

El siguiente ejemplo es un método para la variable vCuadrados. El método de objeto asigna la suma de cuadrados de una serie de datos a vCuadrados. La variable vCuadrados se imprime en la última ruptura del informe:

```
vCuadrados:=Sum squares([Tabla1]SeriesDatos)
```

El siguiente método se llama para imprimir los registros en la selección y activar el proceso de ruptura:

```
ALL RECORDS([Tabla1])
ORDER BY([Tabla1];[Tabla1]SeriesDatos;>)
BREAK LEVEL(1)
ACCUMULATE([Tabla1]SeriesDatos)
OUTPUT FORM([Tabla1];"Imprimir formulario")
PRINT SELECTION([Tabla1])
```

Nota: *el parámetro del comando* ***BREAK LEVEL*** *debe ser igual al número de rupturas en su informe. Para mayor información sobre el proceso de rupturas, consulte* ***Impresión****.*

Ejemplo 2

Este ejemplo permite obtener la suma de cuadrados de los valores ubicados en un array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vSumSquares:=Sum squares($ArrGrades)
```

Variance

Variance (series) -> Resultado

Parámetro	Tipo		Descripción
series	Campo, Array	➡	Datos para los cuales se devuelve la varianza
Resultado	Real	⮌	Varianza de series

Descripción

Variance devuelve la varianza para series. Si series es un campo indexado, el índice se utiliza para calcular la varianza.
Puede pasar en series un array (de una o dos dimensiones). En este caso, el array debe ser de tipo Entero, Entero largo o Real.
La varianza de un conjunto de valores es el promedio de los cuadrados de las desviaciones estándar. La varianza media mide la dispersión de valores alrededor de la media. 4D utiliza la siguiente fórmula de varianza:

Varianza(x) = Sum (x-m)*(x-m)/(n-1)
m = Media
n = Número de valores

Si los valores no se consideran una muestra, multiplique el valor devuelto por **Variance** por (n-1)/n.

Ejemplo 1

El siguiente ejemplo es un método de objeto para la variable var. El método de objeto asigna la suma de cuadrados de una serie de datos a var:

```
var:=Variance(Estudiantes]Notas)
```

El siguiente método se llama para imprimir los registros en la selección y activar el proceso de ruptura:

```
ALL RECORDS([Estudiantes])
ORDER BY([Estudiantes];[Estudiantes]Clase;>)
BREAK LEVEL(1)
ACCUMULATE([Estudiantes]Notas)
OUTPUT FORM([Estudiantes];"Imprimir formulario")
PRINT SELECTION([Estudiantes])
```

Nota: el parámetro del comando **BREAK LEVEL** debe ser igual al número de rupturas en su informe. Para mayor información sobre el proceso de rupturas, consulte **Impresión**.

Ejemplo 2

Este ejemplo permite obtener la varianza de valores ubicados en un array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vVariance:=Variance($ArrGrades)
```

Funciones matemáticas

- *Abs*
- *Arctan*
- *Cos*
- *Dec*
- *Euro converter*
- *Exp*
- *Int*
- *Log*
- *Mod*
- *Random*
- *Round*
- *SET REAL COMPARISON LEVEL*
- *Sin*
- *Square root*
- *Tan*
- *Trunc*

Abs

Abs (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	→	Número del cual quiere obtener el valor absoluto
Resultado	Real	↩	Valor absoluto de número

Descripción

Abs devuelve el valor absoluto (positivo y sin signo) de número. Si número es negativo, se devuelve su valor positivo. Si número es positivo, se devuelve igual.

Ejemplo

El siguiente ejemplo devuelve el valor absoluto de –10.3, que es 10.3:

```
vlVector:=Abs(-10.3)
```

Arctan

Arctan (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	→	Tangente para la cual calcular el ángulo
Resultado	Real	↩	Ángulo en radianes

Descripción

Arctan devuelve el ángulo, expresado en radianes, de la tangente número.

Nota: 4D ofrece las constantes predefinidas Pi, **Degree,** y **Radian**. Pi devuelve el número Pi (3.14159...), **Degree** devuelve el valor en radianes de un grado (0.01745...) y **Radian** devuelve el valor en grados de un radián (57.29577...).

Ejemplo

El siguiente ejemplo muestra el valor de Pi:

```
ALERT("Pi es igual a: "+String(4*Arctan(1)))
```

Cos

Cos (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	⇒	Número, en radianes, cuyo coseno se devuelve
Resultado	Real	⇐	Coseno del número

Descripción

Cos *devuelve el coseno del número, donde número se expresa en radianes.*

Nota: *4D ofrece las constantes predefinidas Pi,* ***Degree,*** *y* ***Radian****. Pi devuelve el número Pi (3.14159...),* ***Degree*** *devuelve el valor en radianes de un grado (0.01745...), y* ***Radian*** *devuelve el valor en grados de un radián (57.29577...).*

Dec

Dec (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	→	Número cuya parte decimal se devuelve
Resultado	Real	↩	Parte decimal de número

Descripción

Dec devuelve la parte decimal de número. El valor devuelto siempre es positivo o cero.

Ejemplo

El siguiente ejemplo utiliza un valor monetario expresado como un número real, y extrae la parte de dólares y la parte de centavos. SivrCantidad es 7.31, luego vlDolares vale 7 y vlCents 31:

```
vlDolares:=Int(vrCantidad) ` Obtener los dólares
vlCents:=Dec(vrCantidad)*100 ` Obtener la parte decimal
```

Euro converter

Euro converter (valor ; deMoneda ; aMoneda) -> Resultado

Parámetro	Tipo		Descripción
valor	Real	➡	Valor a convertir
deMoneda	Cadena	➡	Código de la moneda en que está expresado el valor
aMoneda	Cadena	➡	Código de la moneda a la que debe convertirse el valor
Resultado	Real	↩	Valor convertido

Descripción

*El comando **Euro converter** permite efectuar todo tipo de conversión de valores entre las diferentes monedas de países que pertenecen a la "Zona Euro" y al Euro mismo.*

Puede convertir:

- *una moneda nacional en Euros,*
- *Euros en una moneda nacional,*
- *una moneda nacional en otra moneda nacional. En este caso, la conversión se calcula por intermedio del Euro, como se especifica en la reglamentación Europea. Por ejemplo, para convertir Francos belgas en Marcos alemanés, 4D efectuará las siguientes conversiones: Francos belgas -> Euros -> Marcos alemanés.*

Pase en el primer parámetro el valor a convertir.
El segundo parámetro indica el código de la moneda en el cual el valor está expresado.
El tercer parámetro indica el código de la moneda en el cual el valor será convertido.

Para especificar un código de moneda, 4D ofrece las siguientes constantes predefinidas, ubicadas en el tema "Monedas Euro":

Constante	Tipo	Valor
Austrian Schilling	Cadena	ATS
Belgian Franc	Cadena	BEF
Deutsche Mark	Cadena	DEM
Euro	Cadena	EUR
Finnish Markka	Cadena	FIM
French Franc	Cadena	FRF
Greek Drachma	Cadena	GRD
Irish Pound	Cadena	IEP
Italian Lira	Cadena	ITL
Luxembourg Franc	Cadena	LUF
Netherlands Guilder	Cadena	NLG
Portuguese Escudo	Cadena	PTE
Spanish Peseta	Cadena	ESP

Si es necesario, 4D redondea automáticamente el resultado de la conversión y conserva 2 decimales, excepto para conversiones a Liras italianas, Francos Belgas, Francos de Luxemburgo y Pesetas españolas, para las cuales 4D conserva 0 decimales (el resultado es un número entero).

Las tasas de conversión entre el Euro y 12 de las monedas de los países miembros son las siguientes:

Moneda	Valor para 1 Euro
Chelines austriacos	13.7603
Francos belgas	40.3399
Marco alemán	1.95583
Marco finlandés	5.94573
Franco Francés	6.55957
Dracma Griega	340.750
Libra irlandesa	0.787564
Lira italiana	1936.27
Franco luxemburgues	40.3399
Florin neerlandés	2.20371
Escudo portugués	200.482
Peseta española	166.386

Ejemplo

Estos son algunos ejemplos de conversiones que pueden realizarse con este comando:

```
$valor:=10000  `Valor expresado en Francos franceses
  `Convertir el valor a Euros
$EnEuros:=Euro converter($valor;French Franc;Euro)
  `Convertir el valor a Liras Italianas
$EnLiras:=Euro converter($valor;French Franc;Italian Lira)
```

Exp

Exp (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	⇒	Número a evaluar
Resultado	Real	⇐	Exponencial del número

Descripción

Exp devuelve el exponencial (e = 2.71828...) de número. **Exp** es la función inversa de **Log**.
Nota: 4D ofrece la constante predefinida e number (2.71828...).

Ejemplo

El siguiente ejemplo asigna el exponencial de 1 a vrE (el log de vrE es 1):

```
vrE:=Exp(1) ` vrE vale 2.17828...
```

Int

Int (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	→	Número cuya parte entera se devuelve
Resultado	Real	↩	Parte entera de número

Descripción

Int devuelve la parte entera de número, redondeando al entero inferior.

Ejemplo

El siguiente ejemplo ilustra el funcionamiento de **Int** para números positivos y negativos. Note que la porción decimal del número se elimina:

```
vlResult:=Int(123.4) ` vlResult vale 123
vlResult:=Int(-123.4) ` vlResult vale –124
```

Log

Log (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	→	Número para el cual devolver el logaritmo
Resultado	Real	←	Logaritmo del número

Descripción

Log devuelve el logaritmo neperiano de número. **Log** es la función inversa de **Exp**.
Nota: 4D ofrece la constante predefinida e number (2.71828...).

Ejemplo

La siguiente línea muestra 1:

```
ALERT(String(Log(Exp(1))))
```

Mod

Mod (número1 ; número2) -> Resultado

Parámetro	Tipo		Descripción
número1	Entero largo	➡	Número a dividir
número2	Entero largo	➡	Número divisor
Resultado	Real	⮌	Devuelve el resto de la división

Descripción

*El comando **Mod** devuelve el resto de la división entera de número1 entre número2.*

Notas:

*• **Mod** acepta expresiones de tipo Entero, Entero largo y Reales. Sin embargo, si número1 o número2 son números reales, los números primeros son redondeados y luego se calcula **Mod**.*

*• Sea cuidadoso cuando utilice **Mod** con números reales de gran tamaño (sobre 2^31), ya que en este caso, su operación podría alcanzar los límites de las capacidades de cálculo de los procesadores estándar.*

*Igualmente puede utilizar el operador % para calcular el resto (ver **Operadores numéricos**).*

***Advertencia:** el operador % devuelve resultados válidos con expresiones de tipo Entero y Entero largo. Para calcular el módulo de valores reales, debe utilizar el comando **Mod**.*

Ejemplo

*El siguiente ejemplo ilustra el funcionamiento de **Mod** con diferentes argumentos. Cada línea asigna un número a la variable vlResult. Los comentarios describen los resultados:*

```
vlResult:=Mod(3;2)  ` vlResult vale 1
vlResult:=Mod(4;2)  ` vlResult vale 0
vlResult:=Mod(3.5;2)  ` vlResult vale 0
```

Random

Random -> Resultado

Parámetro	Tipo	Descripción
Resultado	Entero largo	Número aleatorio

Descripción

Random devuelve un valor entero aleatorio entre 0 y 32 767 (inclusive).
Para definir un rango de enteros, utilice esta fórmula:

```
(Random%(vEnd-vStart+1))+vStart
```

El valor vStart es el primer número del intervalo, y el valor vEnd es el último.

Ejemplo

El siguiente ejemplo asigna un valor aleatorio entre 10 y 30 a la variable vlResult:

```
vlResult:=(Random%21)+10
```

Round

Round (redond ; decimales) -> Resultado

Parámetro	Tipo		Descripción
redond	Real	➡	Número a redondear
decimales	Entero largo	➡	Número de lugares decimales a redondear
Resultado	Real	⮌	Número redondeado para el número de lugares decimales especificado por decimales

Descripción

Round devuelve número redondeado al número de decimales especificado por decimales.

Si decimales es positivo, se redondea la parte decimal de número. Si decimales es negativo, se redondea la parte entera de número.

Si la cifra después del número de decimales definido por decimales está entre 5 y 9, redond redondea al valor superior si el número es positivo, y hacia el valor inferior si el número es negativo. Si el digito después de decimales está entre 0 y 4, **Round** redondea hacia cero.

Ejemplo

El siguiente ejemplo ilustra cómo **Redondeo** funciona con diferentes argumentos. Cada línea asigna un número a la variable vlResult. Los comentarios describen los resultados:

```
vlResult:=Round(16.857;2) ` vlResult vale 16.86
vlResult:=Round(32345.67;-3) ` vlResult vale 32000
vlResult:=Round(29.8725;3) ` vlResult vale 29.873
vlResult:=Round(-1.5;0) ` vlResult vale –2
```

SET REAL COMPARISON LEVEL

SET REAL COMPARISON LEVEL (epsilon)

Parámetro	Tipo		Descripción
epsilon	Real	➡	Valor epsilon para las comparaciones de igualdad de los reales

Descripción

El comando ***SET REAL COMPARISON LEVEL*** *define el valor epsilon utilizado por 4D para hacer comparaciones de igualdad de valores y expresiones de tipo real.*

Un ordenador siempre realiza cálculos aproximativos sobre reales; por lo tanto, las pruebas de igualdad de valores reales deben tener en cuenta esta aproximación. 4D hace esto cuando compara números reales probando si la diferencia entre dos valores es superior o no a un cierto valor. Este valor se llama el ***epsilon*** *y funciona de esta manera:*

Dados dos números reales a y b, si ***Abs(a-b)*** *es mayor al epsilon, los números son considerados como diferentes; de lo contrario, los números son considerados iguales.*

Por defecto, 4D, define el valor epsilon en 10 a la potencia menos 6 (10^-6). Por favor note que el valor epsilon siempre debe ser positivo. Ejemplos:

- *0.00001=0.00002 devuelve Falso, porque la diferencia 0.00001 es mayor que 10^-6.*
- *0.000001=0.000002 devuelve Verdadero, porque la diferencia 0.000001 no es mayor que 10^-6.*
- *0.000001=0.000003 devuelve Falso, porque la diferencia 0.000002 es mayor que 10^-6.*

Utilizando ***SET REAL COMPARISON LEVEL****, puede aumentar o reducir el valor epsilon, en función de sus necesidades.*

Advertencia: *generalmente, no necesitará utilizar este comando para modificar el valor epsilon por defecto.*

IMPORTANTE: *cambiar el epsilon sólo afecta la comparación de igualdad de reales. No tiene efecto en los otros cálculos y visualizaciones de valores reales.*

Nota: *el comando* ***SET REAL COMPARISON LEVEL*** *no tiene efecto en las búsquedas y ordenaciones efectuadas con los campos de tipo real. Eso aplica únicamente al lenguaje de 4D.*

Sin

Sin (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	➡	Número, en radianes, cuyo seno se devuelve
Resultado	Real	⮌	Seno del número

Descripción

Sin devuelve el seno del *número*, donde *número* se expresa en radianes.

Nota: 4D ofrece las constantes predefinidas Pi, **Degree,** y **Radian**. Pi devuelve el número Pi (3.14159...), **Degree** devuelve el valor en radianes de un grado (0.01745...), y **Radian** devuelve el valor en grados de un radián (57.29577...).

Square root

Square root (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	➡	Número al que se le va a calcular la raíz cuadrada
Resultado	Real	⮌	Raíz cuadrada del número

Descripción

Square root devuelve la raíz cuadrada de *número*.

Ejemplo 1

La línea:

```
$vrRaizDeDos :=Square root(2)
```

asigna el valor *1.414213562373* a la variable *$vrRaizDeDos*.

Ejemplo 2

El siguiente método devuelve la hipotenusa del triángulo cuyos dos lados son pasados como parámetros:

```
` Método Hipotenusa
` Hipotenusa( real ; real ) -> real
` Hipotenusa( ladoA ; ladoB ) -> Hipotenusa
C_REAL($0;$1;$2)
$0:=Square root(($1^2)+($2^2))
```

Por ejemplo, Hipotenusa (4;3) devuelve 5.

Tan

Tan (Numero) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	⇒	Número, en radianes, cuya tangente se devuelve
Resultado	Real	⇐	Tangente del número

Descripción

Tan devuelve la tangente del *número*, donde *número* se expresa en radianes.

Nota: 4D ofrece las constantes predefinidas *Pi*, **Degree,** y **Radian**. *Pi* devuelve el número Pi (3.14159...), **Degree** devuelve el valor en radianes de un grado (0.01745...), y **Radian** devuelve el valor en grados de un radián (57.29577...).

Trunc

Trunc (Numero ; decimales) -> Resultado

Parámetro	Tipo		Descripción
Numero	Real	➡	Número a truncar
decimales	Entero largo	➡	Número de lugares decimales a conservar
Resultado	Real	⮌	Número truncado a partir del número decimales especificado por decimales

Descripción

Trunc devuelve *número* con su parte decimal truncada a partir del número de decimales especificado por *decimales*. **Trunc** siempre redondea hacia el valor inferior.

Si *decimales* es positivo, *número* se trunca por la parte decimal. Si *decimales* es negativo, el truncamiento se hace sobre la parte entera del número.

Ejemplo

El siguiente ejemplo ilustra la manera cómo **Trunc** funciona con diferentes argumentos. Cada línea asigna un número a la variable *vlResult*. Los comentarios describen los resultados:

```
vlResult:=Trunc(216.897;1)  ` vlResult vale 216.8
vlResult:=Trunc(216.897;-1)  ` vlResult vale 210
vlResult:=Trunc(-216.897;1)  ` vlResult vale –216.9
vlResult:=Trunc(-216.897;-1)  ` vlResult vale –220
```

Gestión de la caché

---

*4D integra un esquema de caché de datos incorporado para acelerar las operaciones I/O. El hecho de que las modificaciones de datos estén por momentos, presentes en la caché de datos y no en el disco es transparente para su codificación. Por ejemplo, si llama al comando **QUERY**, el motor de 4D integra los datos presentes en la caché para efectuar la operación.*

*El administrador de la caché de la base de datos ha sido completamente reescrito en 4D v16, lo que le permite tomar ventaja de los entornos de 64 bits. Automáticamente activado y optimizado, sin embargo, se puede configurar o analizar de forma dinámica con los comandos de este tema. Además, se implementa un sistema de prioridad personalizable. Para más información, consulte la sección **Gestión de prioridades en la caché de la base**.*

*Tenga en cuenta que el selector Cache flush periodicity se puede utilizar con los comandos **SET DATABASE PARAMETER** y **Get database parameter**.*

- *Gestión de prioridades en la caché de la base*
- *ADJUST BLOBS CACHE PRIORITY*
- *ADJUST INDEX CACHE PRIORITY*
- *ADJUST TABLE CACHE PRIORITY*
- *Cache info*
- *FLUSH CACHE*
- *Get adjusted blobs cache priority*
- *Get adjusted index cache priority*
- *Get adjusted table cache priority*
- *Get cache size*
- *GET MEMORY STATISTICS*
- *SET BLOBS CACHE PRIORITY*
- *SET CACHE SIZE*
- *SET INDEX CACHE PRIORITY*
- *SET TABLE CACHE PRIORITY*

Gestión de prioridades en la caché de la base

En las versiones 4D de 64 bits, la caché de la base incluye un mecanismo automático de gestión de prioridades que ofrece un alto nivel de eficiencia y rendimiento para el acceso a los datos. Gracias a este mecanismo, cuando se necesita espacio para cargar nuevos datos en la caché, los datos en caché de baja prioridad se liberan primero, mientras que los datos en caché de mayor prioridad permanecen cargados.

Este mecanismo es totalmente automático y por lo general, no tendrá que preocuparse por él. Sin embargo, para casos específicos se puede personalizar mediante un conjunto de comandos dedicados, lo que permite cambiar la prioridad de los objetos durante todo el tiempo que se ejecuta la base o temporalmente para el proceso actual. Tenga en cuenta que estos comandos deben utilizarse con cuidado ya que afectan al rendimiento de la base.

Mecanismo interno de gestión de prioridades

Además de la edad y la frecuencia, el gestor de memoria caché selecciona los datos para eliminarlos de la memoria caché cuando sea necesario utilizando un sistema de prioridad. Los tres tipos de objetos que se pueden cargar en la memoria caché tienen una prioridad diferente:

- ***tablas**: todos los datos de campo estándar (numéricos, fechas...), excluyendo blobs (ver abajo). El valor de prioridad por defecto es medio*
- ***blobs**: todos los datos de campo binario (texto, imagen, objeto y blob) almacenados en el archivo de datos. La prioridad predeterminada es la más baja.*
- ***índices**: todos los índices de campo, incluidos los índices de palabras clave e índices compuestos. Dado que los índices se acceden con frecuencia, tienen un estado especial en la caché. El valor de prioridad por defecto es el más alto*

Las prioridades predeterminadas suelen ofrecer el mejor rendimiento. Sin embargo, para casos específicos puede personalizar las prioridades de la caché utilizando dos conjuntos de comandos 4D:

- *Los comandos que cambian las prioridades para toda la sesión y todos los procesos: **SET TABLE CACHE PRIORITY**, **SET INDEX CACHE PRIORITY**, y **SET BLOBS CACHE PRIORITY**. Estos comandos deben ser utilizados en un método de base de inicio..*
- *Los comandos que cambian las prioridades solamente para el proceso actual: **ADJUST TABLE CACHE PRIORITY**, **ADJUST INDEX CACHE PRIORITY** y **ADJUST BLOBS CACHE PRIORITY**. Utilice estos comandos para mejorar el rendimiento de una operación temporal en su base de datos y volver a las prioridades iniciales una vez finalizada la operación.*

Estos comandos sólo están disponibles en los siguientes contextos:

- *versiones 4D 64 bits*
- *4D Server o 4D en modo local*

ADJUST BLOBS CACHE PRIORITY

ADJUST BLOBS CACHE PRIORITY (tabla ; prioridad)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla cuyo valor de prioridad de datos "blobs" tiene que ser ajustado
prioridad	Entero largo	➡	Valor de prioridad de la caché para los BLOBs en la tabla

Modo experto

Este comando está reservado para necesidades específicas. Debe utilizarse con cuidado, ya que puede afectar el rendimiento de la base.

Descripción

*El comando **ADJUST BLOBS CACHE PRIORITY** modifica la prioridad de los datos "blobs" relacionados con tabla en la caché para el proceso actual. Una llamada a este comando reemplaza toda prioridad previamente ajustado a través del mismo comando en el mismo proceso. Este comando ajusta la prioridad para una necesidad temporal, por ejemplo durante una búsqueda o una importación.*

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); No se puede utilizar en modo remoto 4D.*

Los tipos de campos de datos "Blobs" incluyen BLOB, texto, imagen y objeto. Este comando maneja la prioridad para tales datos cuando se almacenan en el archivo de datos únicamente.
*La prioridad para los campos de tipo escalar (como los tipos fecha, número o cadenas) es ajustada por el comando **ADJUST TABLE CACHE PRIORITY**.*

*Pase en prioridad una de las siguientes constantes del tema "**Gestión**":*

Constante	Tipo	Valor	Comentario
Cache priority high	Entero largo	1000	
Cache priority low	Entero largo	-900	
Cache priority normal	Entero largo	0	Define la prioridad de la caché a su valor por defecto
Cache priority very high	Entero largo	30000	
Cache priority very low	Entero largo	-1	

Ejemplo

Usted desea cambiar temporalmente la prioridad de la caché de los campos de texto de la tabla [Docs] almacenados en el archivo de datos al ejecutar una búsqueda secuencial:

```
ADJUST BLOBS CACHE PRIORITY([Docs];Cache priority very high)
QUERY([Docs];[Docs]Author#"A@") // búsqueda secuencial de un campo no indexado
 //... ejecutar varias otras búsquedas u ordenaciones en la misma tabla
 // al terminar, volver a la prioridad de caché normal
ADJUST BLOBS CACHE PRIORITY([Docs];Cache priority normal)
```

ADJUST INDEX CACHE PRIORITY

ADJUST INDEX CACHE PRIORITY (campo ; prioridad)

Parámetro	Tipo		Descripción
campo	Campo	➡	Campo cuyo valor de prioridad de índice(s) debe ajustarse
prioridad	Entero largo	➡	Valor de prioridad de la caché para los índices de campo

Modo experto

Este comando está reservado para necesidades específicas. Debe utilizarse con cuidado, ya que puede afectar el rendimiento de la base.

Descripción

*El comando **ADJUST INDEX CACHE PRIORITY** modifica el valor de prioridad de los índices relacionados con campo en la caché para el proceso actual. Una llamada a este comando reemplaza cualquier valor de prioridad previamente ajustado a través del mismo comando en el mismo proceso. Este comando ajusta la prioridad para necesidades temporales, por ejemplo durante una búsqueda o una importación.*

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); No se puede utilizar en modo remoto 4D.*

Este comando controla la prioridad para todos los índices relacionados con campo, incluyendo índices de palabras claves. Sin embargo, no cambia la prioridad de los índices compuestos.

*Pase en prioridad una de las siguientes constantes del tema "**Gestión**":*

Constante	Comentario
Cache priority low	
Cache priority very low	
Cache priority normal	Define la prioridad de la caché a su valor por defecto
Cache priority high	
Cache priority very high	

Ejemplo

Usted desea cambiar temporalmente la prioridad de la caché para el índice de campo [Docs]Comments:

```
ADJUST INDEX CACHE PRIORITY([Docs]Comments;Cache priority very high)
QUERY([Docs];[Docs]Comments%"Extra") // búsqueda de un campo indexado
 //... realización de otras búsquedas u ordenaciones en la misma tabla
 // al terminar, volver a la prioridad de caché normal
ADJUST INDEX CACHE PRIORITY([Docs]Comments;Cache priority normal)
```

ADJUST TABLE CACHE PRIORITY

ADJUST TABLE CACHE PRIORITY (tabla ; prioridad)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla cuyo valor de prioridad de datos escalar tiene que ser ajustado
prioridad	Entero largo	➡	Valor de prioridad de la caché para la tabla

Modo experto

Este comando está reservado para necesidades específicas. Debe utilizarse con cuidado, ya que puede afectar el rendimiento de la base.

Descripción

*El comando **ADJUST TABLE CACHE PRIORITY** modifica el valor de prioridad de los datos relacionados con tabla en la caché para el proceso actual.*
Una llamada a este comando reemplaza cualquier valor de prioridad previamente ajustado a través del mismo comando en el mismo proceso. Este comando ajusta la prioridad de las necesidades temporales, por ejemplo durante una búsqueda o una importación.

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); No se puede utilizar en modo remoto 4D.*

*Este comando controla la prioridad de los datos sólo en los campos escalares (tipos fecha, número o cadena). La prioridad para campos de tipo binario (Blobs, textos, imágenes y objetos) es manejada por el comando **ADJUST BLOBS CACHE PRIORITY**.*

*Pase en prioridad una de las siguientes constantes del tema "**Gestión**":*

Constante	Comentario
Cache priority low	
Cache priority very low	
Cache priority normal	Define la prioridad de la caché a su valor por defecto
Cache priority high	
Cache priority very high	

Ejemplo

Usted desea cambiar temporalmente la prioridad de la caché de los campos escalares [Docs]:

```
ADJUST TABLE CACHE PRIORITY([Docs];Cache priority low)
 // ... hacer alguna operación específica
ADJUST TABLE CACHE PRIORITY([Docs];Cache priority normal)
```

Cache info

Cache info {(dbFilter)} -> Resultado

Parámetro	Tipo		Descripción
dbFilter	Objeto	➡	Define la lista de atributos a devolver (filtrada por DB)
Resultado	Objeto	↩	Información sobre la caché

64 bits

Este comando se basa en una arquitectura interna que sólo está disponible en versiones 4D de 64 bits. Este comando devolverá un error si se llama desde una versión de 32 bits de 4D.

Descripción

*El comando **Cache info** devuelve un objeto que contiene la información detallada sobre el contenido actual de la caché (memoria utilizada, tablas e índices cargados, etc.)*

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); no debe ser utilizado con 4D en modo remoto.*

Por defecto, la información devuelta se refiere solamente a la base en ejecución. El parámetro objeto opcional dbFilter le permite especificar el alcance del comando:

- *pase el atributo "dbFilter" con el valor "All" para obtener la información acerca de la caché de todas las bases lanzadas, incluidos los componentes.*
- *pase el atributo "dbFilter" con un valor "" (cadena vacía) para obtener información sobre la única base de datos actual (equivale a omitir el parámetro dbFilter).*

*El comando **Cache info** devuelve un único objeto que contiene toda la información relevante acerca de la caché. El objeto devuelto tiene la siguiente estructura básica:*

```
{
    "maxMem": Maximum cache size (real),
    "usedMem": Current cache size (real),
    "objects": [...] Array of objects currently loaded in cache
}
```

Los elementos del array objetos son objetos raíz (tablas, índices, Blobs, etc.) que estén cargados en la memoria caché. Cada elemento contiene atributos específicos que describen su estado actual. Para más información acerca de la interpretación avanzada de estos datos, contacte a su departamento de servicio técnico.

Ejemplo

Quiere obtener la información de la caché para la base de datos actual:

```
C_OBJECT($cache)
$cache:=Cache info
```

Usted quiere obtener información de la caché de la base de datos y todos los componentes abiertos:

```
C_OBJECT($dbFilter)
OB SET($dbFilter;"dbFilter";"All")
$cache:=Cache info($dbFilter)
```

FLUSH CACHE

FLUSH CACHE {(tam | *)}

Parámetro	Tipo		Descripción
tam | *	Real, Operador	➡	* para liberar la memoria caché completamente, o número de bytes a liberar en la caché

Descripción

*El comando **FLUSH CACHE** guarda inmediatamente los buffers de datos en el disco. Todos los cambios realizados a la base se guardan en el disco.*

Por defecto, este comando no afecta el contenido actual de la caché, esto significa que los datos se siguen utilizando en acceso en lectura posteriores. Opcionalmente, se puede pasar un parámetro a modificar su contenido:

- *pase * para guardar la caché y liberar toda la memoria caché.*
- *pase un valor tam para guardar la caché y liberar sólo el número tam de bytes de la caché.*

***Nota:** pasar un parámetro a este comando es para propósitos de prueba. Por motivos de rendimiento, no se recomienda liberar la caché en el entorno de producción.*

*Normalmente, no es necesario llamar a este comando, ya que 4D guarda las modificaciones de datos regularmente. La opción **Vaciar la caché cada X segundos (minutos)** en **Página Base de datos/Memoria** de las Propiedades de la base, que especifica con qué frecuencia guardar, se suele utilizar para controlar el vaciado de la caché. Recomendamos utilizar el valor por defecto de 20 segundos. Tenga en cuenta también que el parámetro Cache flush periodicity se puede ajustar y leer utilizando los comandos **SET DATABASE PARAMETER** y **Get database parameter**.*

Get adjusted blobs cache priority

Get adjusted blobs cache priority (tabla) -> Resultado

Parámetro	*Tipo*		*Descripción*
tabla	*Tabla*	➡	*Tabla para la cual el valor de prioridad de los "blobs" tiene que ser recuperado*
Resultado	*Entero largo*	⮌	*Valor de prioridad actual para los campos "blobs"*

Descripción

*El comando **Get adjusted blobs cache priority** devuelve el valor de prioridad de la caché ajustada actual aplicado en la caché para los datos "blobs" tabla. Este comando sólo es necesario para propósitos de depuración.*

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); No se puede utilizar en modo remoto 4D.*

Get adjusted index cache priority

Get adjusted index cache priority (campo) -> Resultado

Parámetro	*Tipo*		*Descripción*
campo	*Campo*	➡	*Campo para el cual se debe recuperar la prioridad del índice*
Resultado	*Entero largo*	⮌	*Valor de la prioridad actual para los índices*

Descripción

*El comando **Get adjusted index cache priority** devuelve el valor de prioridad de caché ajustado actual aplicado por el gestor de caché para los índices de campo. Este comando sólo es necesario para propósitos de depuración.*

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); No se puede utilizar en modo remoto 4D.*

Get adjusted table cache priority

Get adjusted table cache priority (tabla) -> Resultado

Parámetro	*Tipo*		*Descripción*
tabla	*Tabla*	➡	*Tabla para la cual se debe recuperar el valor de prioridad escalar*
Resultado	*Entero largo*	⮌	*Valor de prioridad actual para campos escalares*

Descripción

El comando ***Get adjusted table cache priority*** *devuelve el valor de prioridad de la caché ajustada aplicado en caché para los datos escalares tabla.. Este comando sólo es necesario para propósitos de depuración.*

Nota: *este comando sólo funciona en modo local (4D Server y 4D); No se puede utilizar en modo remoto 4D.*

Los tipos de campos de datos escalares incluyen campos de tipo fecha/hora, numéricos o cadena.

Get cache size

Get cache size -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Real*		*Tamaño, en bytes, de la caché de la base de datos*

Descripción

*El comando **Get cache size** devuelve, en bytes, el tamaño actual de la caché de la base de datos.*
***Nota:** este comando sólo funciona en modo local (4D Server y 4D); no debe ser utilizado con 4D en modo remoto.*

Ejemplo

*Ver el ejemplo del comando **SET CACHE SIZE**.*

GET MEMORY STATISTICS

GET MEMORY STATISTICS (tipoInfo ; arrayNombres ; arrayValores ; ArrayContador)

Parámetro	Tipo		Descripción
tipoInfo	Entero largo	➡	Selector de información a obtener
arrayNombres	Array texto	⬅	Títulos de la información
arrayValores	Array real	⬅	Valores de la información
ArrayContador	Array real	⬅	Número de objetos respectivos (si disponible)

Descripción

*El comando **GET MEMORY STATISTICS** recupera la información relativa al uso de la caché de datos por 4D.. Esta información puede utilizarse en el análisis del funcionamiento de la aplicación.*

Pase en el parámetro tipoInfo un valor indicando el tipo de información que quiere obtener:

- *1 = Información general en la memoria. Esta información está disponible igualmente vía el explorador de ejecución: tamaño de memoria física, virtual, libre, ocupada, etc.)*
- *2 = Resumen de las estadísticas de ocupación de la caché de la base (4D 32-bits únicamente).*

***Nota de compatibilidad:** el valor 2 es obsoleto para las versiones 4D de 64 bits. Para estas versiones, se recomienda utilizar el comando [#cmd id="1402"/] para obtener lad estadísticas de la caché.*

Después de ejecutar el comando, las estadísticas solicitadas se entregan en los arrays arraysNombres, arraysValores y ArrayContador. Para mayor información sobre la interpretación avanzada de estos datos, contacte al departamento de servicio técnico de 4D.

SET BLOBS CACHE PRIORITY

SET BLOBS CACHE PRIORITY (tabla ; prioridad)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla cuyo valor de prioridad de datos "blobs" tiene que definirse para la sesión
prioridad	Entero largo	➡	Valor de prioridad de la caché para BLOBs en la tabla

Modo experto

Este comando está reservado para necesidades específicas. Debe utilizarse con cuidado, ya que puede afectar el rendimiento de la base.

Descripción

*El comando **SET BLOBS CACHE PRIORITY** define una prioridad específica para los datos "blobs" relacionados con tabla in la caché para todos los procesos de la sesión actual. Este comando debe llamarse en el método base **On Startup** u **On Server Startup**.*

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); No se puede utilizar en modo remoto 4D.*

Los tipos de campos de datos "Blobs" incluyen BLOB, texto, imagen y objeto. Este comando maneja la prioridad para tales datos cuando se almacenan en el archivo de datos únicamente.

*Pase en prioridad una de las siguientes constantes del tema "**Gestión**":*

Constante	Comentario
Cache priority low	
Cache priority very low	
Cache priority normal	Define la prioridad de la caché a su valor por defecto
Cache priority high	
Cache priority very high	

Ejemplo

*En el **Método base On Startup** , usted desea definir una prioridad alta para la tabla [Customer]:*

```
SET BLOBS CACHE PRIORITY([Customer];Cache priority very high)
```

SET CACHE SIZE

SET CACHE SIZE (tam {; libMin})

Parámetro	Tipo		Descripción
tam	Real	➡	Tamaño de la caché de base de datos en bytes
libMin	Real	➡	Número mínimo de bytes a liberar cuando la caché está llena

64 bits

Este comando se basa en una arquitectura interna que sólo está disponible en versiones 4D de 64 bits. Este comando devolverá un error si se llama desde una versión de 32 bits de 4D.

Descripción

*El comando **SET CACHE SIZE** [#descv]define dinámicamente el tamaño de la caché de la base de datos y, opcionalmente, permite definir el tamaño minino en bytes a partir del cual comenzar a liberar memoria[#/descv].*

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); no debe ser utilizado desde un 4D en modo remoto.*

En tam, pase el nuevo tamaño de la memoria caché de la base en bytes. Este nuevo tamaño se aplica de forma dinámica cuando se ejecuta el comando.

En libMin, pase el tamaño mínimo de memoria a liberar de la caché de la base cuando el motor necesita espacio para asignar un objeto a él (valor en bytes). El propósito de esta opción es reducir el número de veces que los datos se liberan de la memoria caché con el fin de obtener un mejor rendimiento. Por defecto, si esta opción no se utiliza, 4D descarga al menos el 10% de la caché cuando se necesita espacio. Si su base de datos funciona con una gran caché, puede ser ventajoso utilizar un tamaño fijo que no dependa del tamaño de la memoria caché. Puede ajustar esta configuración de acuerdo con el tamaño de los bloques de datos que maneja en su base de datos.

Ejemplo

Usted quiere añadir 100 MB al tamaño de la caché de su base actual. Puede escribir:

```
C_REAL($currentCache)
$currentCache:=Get cache size
 // el tamaño de la caché actual, e por ejemplo, 419430400
SET CACHE SIZE($currentCache+10000000)
 // el tamaño de la caché actual ahora es 519430400
```

SET INDEX CACHE PRIORITY

SET INDEX CACHE PRIORITY (campo ; prioridad)

Parámetro	Tipo		Descripción
campo	Campo	➡	Campo cuyo valor de prioridad de índices debe ser configurado para la sesión
prioridad	Entero largo	➡	Valor de prioridad de la caché para el(los) índice(s) de campo

Modo experto

Este comando está reservado para necesidades específicas. Debe utilizarse con cuidado, ya que puede afectar el rendimiento de la base.

Descripción

*El comando **SET INDEX CACHE PRIORITY** define una prioridad específica en los índices relacionados con el campo en caché para todos los procesos de la sesión actual. Este comando debe llamarse en el método base **On Startup** u **On Server Startup**.*

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); No se puede utilizar en 4D modo remoto.*

Este comando maneja la prioridad para todos los índices relacionados con el campo, incluidos los índices de palabras clave (la prioridad de los índices compuestos no se puede personalizar).

*Pase en prioridad una de las siguientes constantes del tema "**Gestión**":*

Constante	Comentario
Cache priority low	
Cache priority very low	
Cache priority normal	Define la prioridad de la caché a su valor por defecto
Cache priority high	
Cache priority very high	

Ejemplo

*En el **Método base On Startup** , usted desea definir una alta prioridad para los índices campo [Cliente]Apellido:*

```
SET INDEX CACHE PRIORITY([Customer]LastName;Cache priority very high)
```

SET TABLE CACHE PRIORITY

SET TABLE CACHE PRIORITY (tabla ; prioridad)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla cuyo valor de prioridad de datos escalares tiene que ser definido para la sesión
prioridad	Entero largo	➡	Valor de prioridad de caché para valores escalares en la tabla

Modo experto

Este comando está reservado para necesidades específicas. Debe utilizarse con cuidado, ya que puede afectar el rendimiento de la base.

Descripción

*El comando **SET TABLE CACHE PRIORITY** define una prioridad específica para los datos relacionados con tabla en caché para todos los procesos de la sesión actual. Este comando debe llamarse en el método base **On Startup** u **On Server Startup**.*

***Nota:** este comando sólo funciona en modo local (4D Server y 4D); No se puede utilizar en 4D modo remoto.*

*Este comando controla la prioridad de los datos sólo en los campos escalares (fecha, número o tipos de cadena). La prioridad para campos de tipo binario (Blobs, textos, imágenes y objetos) es manejada por el comando **SET BLOBS CACHE PRIORITY**.*

Pase en prioridad una de las siguientes constantes del tema:

Constante	Comentario
Cache priority low	
Cache priority very low	
Cache priority normal	Define la prioridad de la caché a su valor por defecto
Cache priority high	
Cache priority very high	

Ejemplo

*En el **Método base On Startup** , usted desea definir una prioridad alta para el dato escalar [Customer]:*

```
SET TABLE CACHE PRIORITY([Customer];Cache priority very high)
```

Gráficos

GRAPH
GRAPH SETTINGS
_o_GRAPH TABLE

GRAPH

GRAPH (grafImagen ; grafNum ; xCategorias {; yValores} {; yValores2 ; ... ; yValoresN})

Parámetro	Tipo		Descripción
grafImagen	Variable imagen	➡	Variable imagen
grafNum	Entero largo, Objeto	➡	Número de tipo de gráfico
xCategorias	Array	➡	Categorías para el eje x
yValores	Array	➡	Valores a representar gráficamente (hasta 8)

Descripción

***Nota de compatibilidad**: a partir de 4D v14 el comando **GRAPH** sólo funciona con una variable imagen como su primer parámetro. La sintaxis obsoleta utilizando un área de gráfico (4D Chart) ya no se soporta.*

*El comando **GRAPH** crea un gráfico en una variable imagen, a partir de los valores provenientes de los arrays. El comando **GRAPH** debe ser ubicado en el método formulario o en un método objeto perteneciente al formulario, o en un método proyecto llamado por uno de estos dos métodos.*

Los gráficos generados por este comando pueden ser dibujados vía el motor de renderización SVG. Se benefician de las funciones de interfaz asociadas a las variables imágenes: menú contextual en modo Aplicación (para permitirle elegir, más particularmente, el formato de visualización), barras de desplazamiento, etc.

***Nota:** SVG (Scalable Vector Graphics) es un formato de archivo gráfico vectorial (extensión .svg). Basado en XML, este formato está ampliamente extendido y puede mostrarse particularmente en navegadores web. Para mayor información, consulte la siguiente dirección: http://www.w3.org.Graphics/SVG/. El comando **SVG EXPORT TO PICTURE** también puede ser utilizado para tomar ventaja del motor SVG integrado.*

En el parámetro grafImagen, pase el nombre de la variable imagen que muestra el gráfico en el formulario.
El segundo parámetro define el tipo de gráfico a utilizar. Hay dos posibilidades:

- *pasar un parámetro grafNum de tipo Entero largo (todas las versiones de 4D): en este caso, debe pasar un número entre 1 y 8. Los diferentes tipos de gráficos se describen en el ejemplo abajo. Después de dibujar un gráfico, puede cambiar el tipo modificando el valor de grafNum y ejecutando nuevamente el comando **GRAPH**. Luego puede modificar ciertas características del gráfico llamando al comando **GRAPH SETTINGS**. Ver el ejemplo 1.*
- *pasar un parámetro grafParams de tipo Objeto (versiones 4D de 64 bits solamente , excepto 4D Server Windows 64 bits): en este caso, debe pasar un objeto que contenga las diversas propiedades del gráfico a definir. Para ello, puede utilizar las constantes que se encuentran en el tema "**Parámetros Gráficos**" (ver más adelante). Esta sintaxis permite definir el tipo de gráfico junto con sus parámetros específicos (leyenda, xmin, etc.) en una sola llamada. Esto permite a los usuarios guardar los gráficos generados como imágenes SVG regulares y hace que sea posible visualizarlos utilizando un navegador estándar como Firefox, Chrome, IE o Safari (los gráficos generados cumplen con la norma SVG implementada en los navegadores). Además, esta sintaxis le permite acceder a varios ajustes adicionales, que le permiten personalizar, por ejemplo, el espaciado entre las barras, los márgenes, colores de la barra, etc. Ver los ejemplos 2, 3 y 4. **Advertencia**: si utiliza esta sintaxis, el comando **GRAPH SETTINGS** NO debe llamarse.*

El parámetro xCategorias define las etiquetas que serán utilizadas por el eje x. (la parte inferior del gráfico). Este dato puede ser de tipo alfa, hora, fecha o numérico. Debe haber el mismo de elementos de array en xCategorias que en cada yValores.

Los datos especificados por yValores son los datos a graficar. Los datos deben ser de tipo numérico. Se puede graficar hasta ocho conjuntos de datos. Los gráficos por sectores sólo representan los primeros yValores.

IDs automáticos

Los IDs específicos se asignan automáticamente a los elementos presentes en el gráfico SVG:

IDs	Descripción
ID_graph_1 to ID_graph_8	Columnas, líneas, áreas...
ID_graph_shadow_1 to ID_graph_shadow_8	Sombras de las columnas, líneas, áreas...
ID_bullet_1 to ID_bullet_8	Puntos (gráficos en líneas y puntos únicamente)
ID_pie_label_1 to ID_pie_label_8	Etiquetas de los sectores (gráficos en sectores únicamente)
ID_legend	Legend
ID_legend_1 to ID_legend_8	Títulos de las leyendas
ID_legend_border	Bordes de las leyendas
ID_legend_border_shadow	Sombras de los bordes de las leyendas
ID_x_values	Valores eje X
ID_y_values	Valores del eje Y
ID_y0_axis	Valores del eje Z
ID_background	Fondo
ID_background_shadow	Fondo
ID_x_grid	Rejilla en el eje X
ID_x_grid_shadow	Sombra de la rejilla en el eje X
ID_y_grid	Rejilla en el eje Y
ID_y_grid_shadow	Sombra de la rejilla en el eje Y

Atributos grafParams

*Cuando se utiliza el parámetro grafParams, se pasa un objeto que contiene las diferentes propiedades del gráfico a definir. Puede utilizar las siguientes constantes, que se encuentran en el tema de constantes "**Parámetros Gráficos**":*

Constante	Tipo	Valor	Comentario
Graph background color	Cadena	graphBackgroundColor	**Valores posibles**: expresión de color compatible con SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414"
Graph background opacity	Cadena	graphBackgroundOpacity	**Valores posibles**:enteros, rango de 0-100 **Valor por defecto**: 100
Graph background shadow color	Cadena	graphBackgroundShadowColor	**Valores posibles**: expresión de color compatible con SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414"
Graph bottom margin	Cadena	bottomMargin	**Valores posibles**: números reales **Valor por defecto**: 12
Graph colors	Cadena	colors	**Valores posibles**: array texto. Los colores para cada serie de gráfico. **Valores por defecto**: Blue-green (#19BAC9), Yellow (#FFC338), Purple (#573E82), Green (#4FA839), Orange (#D95700), Blue (#1D9DF2), Yellow-green (#B5CF32), Red (#D43A26)
Graph column gap	Cadena	columnGap	**Valores posibles**:enteros **Valor por defecto**: 12 Define el espacio entre las barras
Graph column width max	Cadena	columnWidthMax	**Valores posibles**: números reales **Valor por defecto**: 200
Graph column width min	Cadena	columnWidthMin	**Valores posibles**: números reales **Valor por defecto**: 10
Graph default height	Cadena	defaultHeight	**Valores posibles**: números reales **Valor por defecto**: 400. Si graphType=7 (Pie), luego valor por defecto = 600
Graph default width	Cadena	defaultWidth	**Valores posibles**: números reales **Valor por defecto:** 600. Si graphType=7 (Pie), luego valor por defecto = 800
Graph display legend	Cadena	displayLegend	**Valores posibles:** Booleano **Valor por defecto:** True
Graph document background color	Cadena	documentBackgroundColor	**Valores posibles**: expresión color SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414". Cuando un gráfico guardado como imagen SVG se abre en otro lugar, el color de fondo del documento sólo se muestra si el motor de renderizado soporta la norma SVG Tiny 1.2 (soportado por IE, Firefox, pero no en Chrome).
Graph document background opacity	Cadena	documentBackgroundOpacity	**Valores posibles**: entero, rango 0-100 (valor por defecto: 100). Cuando un gráfico guardado como imagen SVG se abre en otro lugar, el color de fondo del documento sólo se muestra si el motor de renderizado soporta la norma SVG Tiny 1.2 (soportado por IE, Firefox, pero no en Chrome).
Graph font color	Cadena	fontColor	**Valores posibles:** expresión color SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414"
Graph font size	Cadena	fontSize	**Valores posibles**: enteros largos **Valor por defecto**: 12. Si graphType=7 (Pie), ver Graph pie font size
Graph left margin	Cadena	leftMargin	**Valores posibles**: números reales **Valor por defecto**: 12
Graph legend font color	Cadena	legendFontColor	**Valores posibles**: expresión de color compatible con SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414"
Graph legend icon gap	Cadena	legendIconGap	**Valores posibles**: números reales **Valor por defecto**: Graph legend icon height/2
Graph legend icon height	Cadena	legendIconHeight	**Valores posibles**: números reales **Valor por defecto**: 20
Graph legend icon width	Cadena	legendIconWidth	**Valores posibles**: números reales **Valor por defecto**: 20
Graph legend labels	Cadena	legendLabels	**Valores posibles:** array texto. Si falta, 4D muestra íconos sin texto.
Graph line width	Cadena	lineWidth	**Valores posibles**: números reales **Valor por defecto**: 2
Graph pie direction	Cadena	pieDirection	**Valores posibles**: 1 o -1 **Valor por defecto**: 1 1 indica la dirección de las agujas del reloj, -1 indica la dirección en sentido antihorario

Constante	Tipo	Valor	Comentario
Graph pie font size	Cadena	pieFontSize	**Valores posibles**: números reales **Valor por defecto**: 16
Graph pie shift	Cadena	pieShift	**Valores posibles**: números reales **Valor por defecto**: 8
Graph pie start angle	Cadena	pieStartAngle	**Valores posibles**: números reales (positivos o negativos) **Valor por defecto**: 0, que representa un ángulo de inicio de 0° (posición hacia arriba) Un valor positivo representa un ángulo relativo a la dirección actual del pie. Un valor negativo representa un ángulo relativo a la dirección opuesta al pie
Graph plot height	Cadena	plotHeight	**Valores posibles**: números reales **Valor por defecto**: 12
Graph plot radius	Cadena	plotRadius	**Valores posibles**: números reales **Valor por defecto**: 12
Graph plot width	Cadena	plotWidth	**Valores posibles**: números reales **Valor por defecto**: 12
Graph right margin	Cadena	rightMargin	**Valores posibles**: números reales **Valor por defecto**: 2
Graph top margin	Cadena	topMargin	**Valores posibles**: números reales **Valor por defecto**: 2
Graph type	Cadena	graphType	**Valores posibles**: enteros largos [1 a 8], donde 1 = barras, 2 = proporcional, 3 = apilados, 4 = líneas, 5 = superficies, 6 = puntos, 7 = pie, 8 = imágenes. **Valor por defecto**: 1 Si es nulo, el gráfico no se dibuja y no se muestra ningún mensaje de error. Si está fuera de rango, el gráfico no se dibuja y se muestra un mensaje de error. Si desea modificar gráficos de tipo imagen (valor=8), debe copiar la carpeta 4D/Resources/GraphTemplates/Graph_8_Pictures/ en la carpeta Resources de su base y realizar las modificaciones necesarias. Los archivos imagen locales se utilizarán en lugar de los archivos 4D. No hay un patrón de nombres de imágenes; 4D ordena los archivos contenidos en la carpeta y le asigna el primer archivo al primer gráfico. Estos archivos pueden ser del tipo SVG o imagen.
Graph xGrid	Cadena	xGrid	**Valores posibles**: Booleano **Valor por defecto**: True. Sólo se utiliza con los tipos proporcionales 4 y 6.
Graph xMax	Cadena	xMax	**Valores posibles**: número, fecha, hora (mismo tipo que el parámetro xLabels). Sólo los valores más altos que xMax se muestran en el gráfico. xMax se utiliza solamente para los tipos de gráficos 4, 5 o 6 si xProp = true y si el tipo xLabels es un número, fecha u hora. Si falta o si xMin>xMax, 4D calcula automáticamente el valor xMax.
Graph xMin	Cadena	xMin	**Valores posibles**: número, fecha, hora (mismo tipo que el parámetro xLabels). Sólo los valores más altos que xMin se muestran en el gráfico. xMin se utiliza solamente para los tipos de gráficos 4, 5 o 6 si xProp = true y si el tipo xLabels es un número, fecha u hora. Si falta o si xMin>xMax, 4D calcula automáticamente el valor xMin.
Graph xProp	Cadena	xProp	**Valores posibles**: Booleano **Valor por defecto:**True True para eje x proporcional; False para eje x normal. xProp se utiliza únicamente para los tipos de gráficos 4, 5 o 6
Graph yGrid	Cadena	yGrid	**Valores posibles**: Booleano **Valor por defecto**: True
Graph yMax	Cadena	yMax	**Valores posibles**: números Si falta, 4D calcula automáticamente el valor yMax.
Graph yMin	Cadena	yMin	**Valores posibles**: números Si falta, 4D calcula automáticamente el valor yMin.

Ejemplo 1

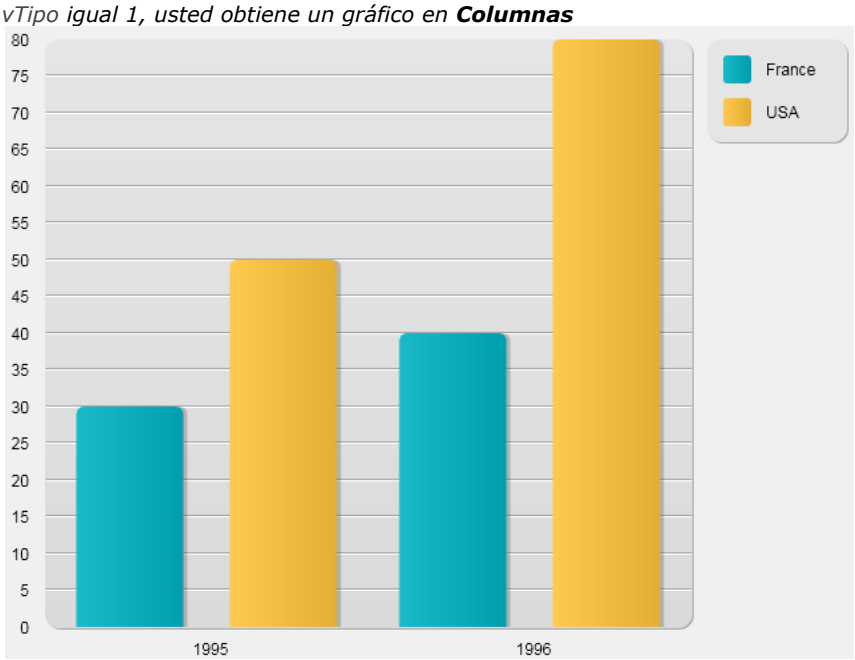
Sintaxis con grafNumber: el siguiente ejemplo muestra los diferentes tipos de gráficos que puede obtener. El código debe ser insertado en un método formulario o método objeto del formulario que contiene la variable imagen. Los datos representados son constantes, generalmente este no es el caso:

```
C_PICTURE(vGraph) //Variable del gráfico
ARRAY TEXT(X;2) //Creación de un array para el eje X
X{1}:="1995" //X Label #1
X{2}:="1996" //X Label #2
ARRAY REAL(A;2) //Creación de un array para el eje Y
A{1}:=30 //Insertar algunos datos
A{2}:=40
```

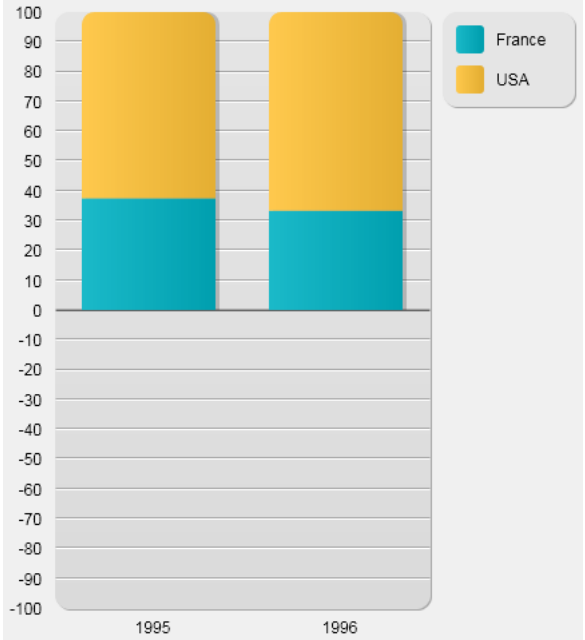
```
ARRAY REAL(B;2) //Creación de un segundo array para el eje Y
B{1}:=50 //Inserción de datos
B{2}:=80
vTipo:=1 //Inicializar tipo de gráfico
GRAPH(vGraph;vTipo;X;A;B) //Dibujar el gráfico
GRAPH SETTINGS(vGraph;0;0;0;0;False;False;True;"France";"USA") //Definición de las leyendas para el gráfico
```

Las siguientes imágenes muestran los gráficos resultantes:

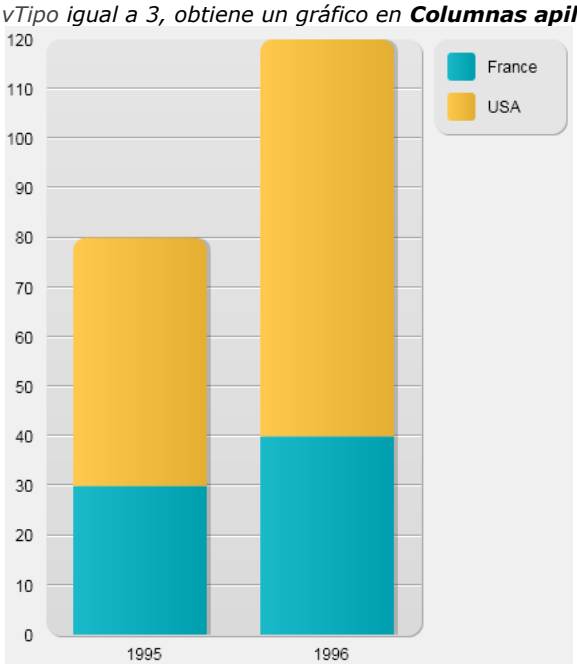
- *vTipo igual 1, usted obtiene un gráfico en* ***Columnas***



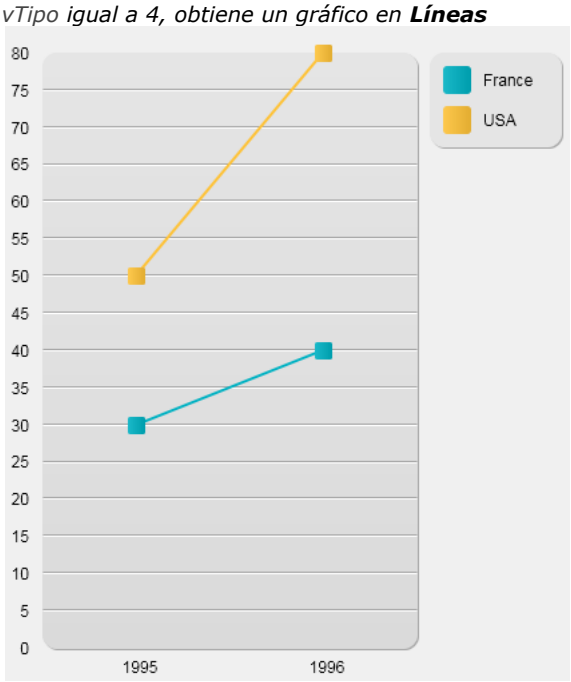
- *vTipo igual a 2, obtiene un gráfico en* ***Columnas proporcionales***



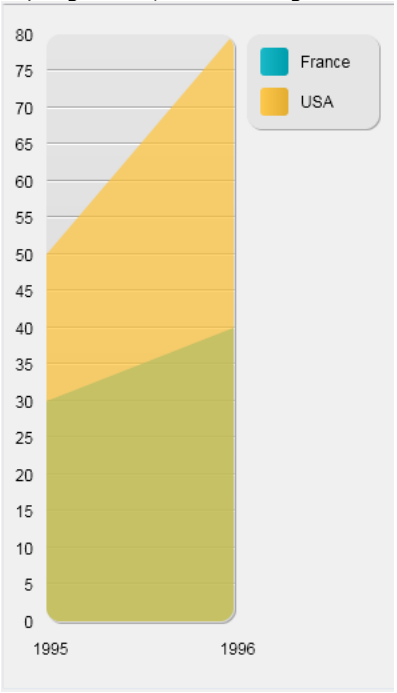
- *vTipo igual a 3, obtiene un gráfico en* ***Columnas apiladas***



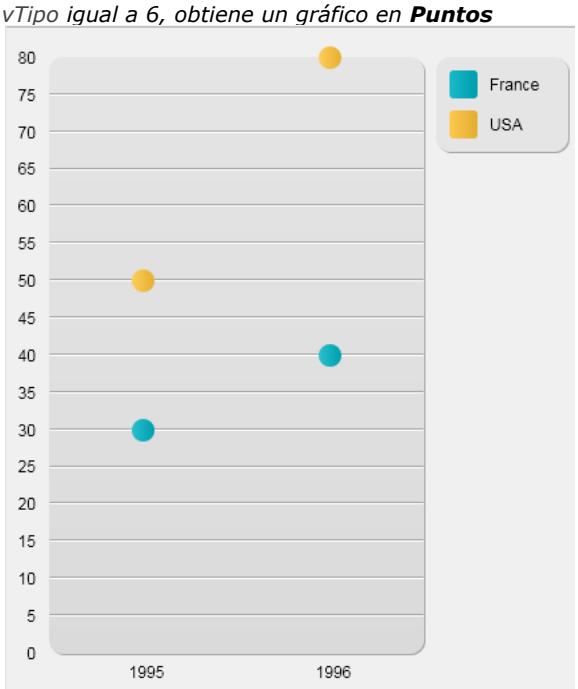
- *vTipo igual a 4, obtiene un gráfico en* ***Líneas***



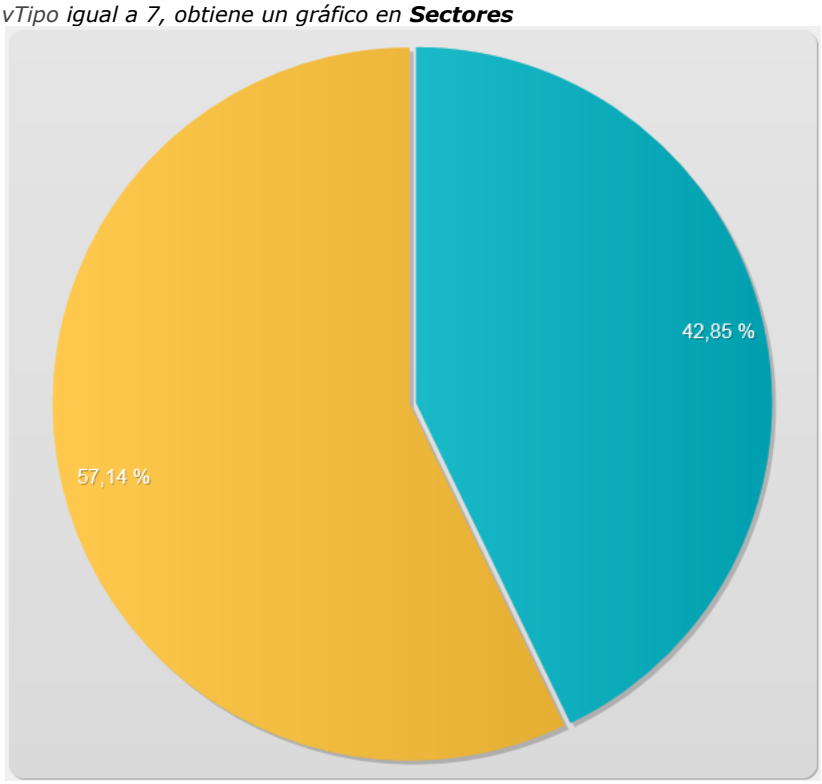
- *vTipo igual a 5, obtiene un gráfico en* ***Áreas***



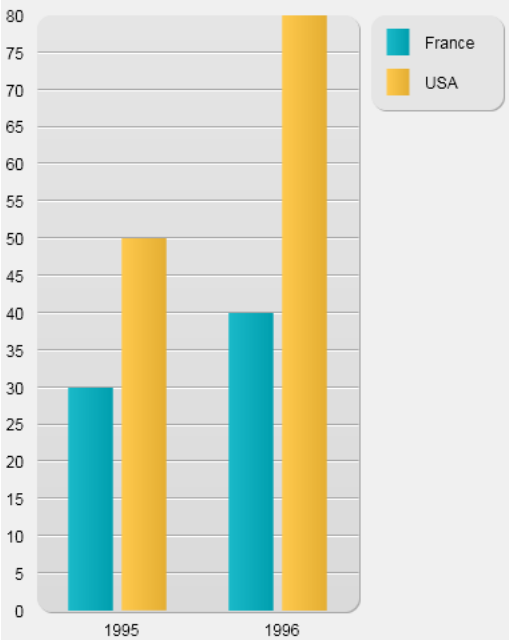
- *vTipo igual a 6, obtiene un gráfico en* ***Puntos***



- *vTipo igual a 7, obtiene un gráfico en* ***Sectores***



- *vTipo igual a 8, obtiene un gráfico en* ***Imágenes***



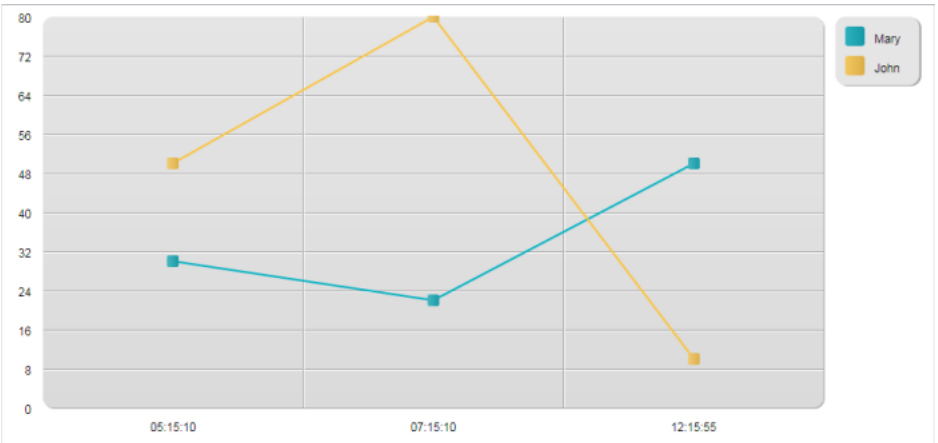
Nota*: las imágenes son rectángulos simples por defecto*

Ejemplo 2

Sintaxis utilizando grafParams: en el siguiente ejemplo, se dibuja un gráfico de línea sencilla basado en los valores de tiempos:

```
C_PICTURE(vGraph) //Variable gráfico
ARRAY TIME(X;3) //Crear array para los ejes x
X{1}:=?05:15:10? //Etiqueta X #1
X{2}:=?07:15:10? //Etiqueta X #2
X{3}:=?12:15:55? //Etiqueta X #3

ARRAY REAL(A;3) //Crear array para eje y
A{1}:=30 //Insertar algunos datos
A{2}:=22
A{3}:=50

ARRAY REAL(B;3) //Crear otro array para eje y
B{1}:=50 //Insertar algunos datos
B{2}:=80
B{3}:=10
```

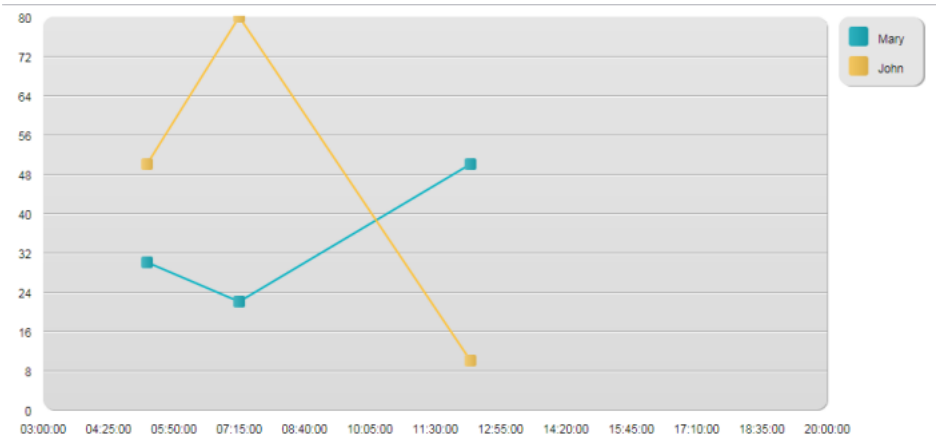
```
C_OBJECT(vSettings) //Inicializar parámetros de gráficos

OB SET(vSettings;Graph type;4) //Tipo de línea

ARRAY TEXT(aLabels;2) //Definir leyendas para el gráfico
aLabels{1}:="Mary"
aLabels{2}:="John"
OB SET ARRAY(vSettings;Graph legend labels;aLabels)

GRAPH(vGraph;vSettings;X;A;B) //Draw graph
```



Ejemplo 3

Con los mismos valores, puede agregar una configuración personalizada para obtener una vista diferente:

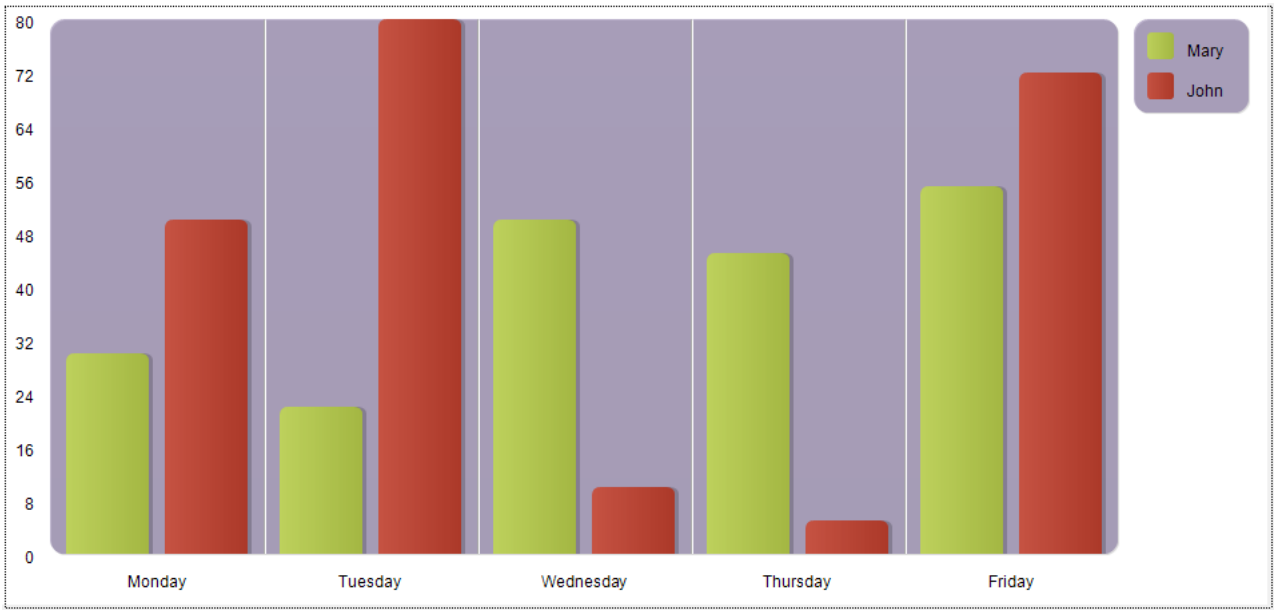
```
C_PICTURE(vGraph) //Variable gráfico
ARRAY TIME(X;3) //Crear un array para el eje x
X{1}:=?05:15:10? //Etiqueta X #1
X{2}:=?07:15:10? //Etiqueta X #2
X{3}:=?12:15:55? //Etiqueta X #3

ARRAY REAL(A;3) //Crea un array para el eje y
A{1}:=30 //Inserta algunos datos
A{2}:=22
A{3}:=50

ARRAY REAL(B;3) //Crea otro array para el eje y
B{1}:=50 //Inserta algunos datos
B{2}:=80
B{3}:=10

C_OBJECT(vSettings) //inicialización de los parámetros del gráfico

OB SET(vSettings;Graph type;4)/Tipo de líneas

ARRAY TEXT(aLabels;2) //definir las leyendas del gráfico
aLabels{1}:="Mary"
aLabels{2}:="John"
OB SET ARRAY(vSettings;Graph legend labels;aLabels)

 //opciones
OB SET(vSettings;Graph xProp;True) //definir proporcional
OB SET(vSettings;Graph xGrid;False) //eliminar la rejilla vertical
OB SET(vSettings;Graph xMin;?03:00:00?) //definir límites
OB SET(vSettings;Graph xMax;?20:00:00?)

GRAPH(vGraph;vSettings;X;A;B) //Dibuja el gráfico
```



Ejemplo 4

En este ejemplo, personalizamos algunos parámetros:

```
C_PICTURE(vGraph) //Variable del gráfico
ARRAY TEXT(X;5) //Creación de un array para el eje x
X{1}:="Monday" //#1 Etiqueta X
X{2}:="Tuesday" //#2 Etiqueta X
X{3}:="Wednesday" //#3 Etiqueta X
X{4}:="Thursday" //#4 Etiqueta X
X{5}:="Friday" //#5 Etiqueta X

ARRAY LONGINT(A;5) //Crear un array para el eje y
A{1}:=30 //Insertar algunos datos
A{2}:=22
A{3}:=50
A{4}:=45
A{5}:=55

ARRAY LONGINT(B;5) //Crear otro array para el eje y
B{1}:=50 //Insertar algunos datos
B{2}:=80
B{3}:=10
B{4}:=5
B{5}:=72

C_OBJECT(vSettings) //inicialización de parámetros del gráfico

OB SET(vSettings;Graph type;1) //tipo Barras

ARRAY TEXT(aLabels;2) //Definir las leyendas para el gráfico
aLabels{1}:="Mary"
aLabels{2}:="John"
OB SET ARRAY(vSettings;Graph legend labels;aLabels)

 //opciones
OB SET(vSettings;Graph yGrid;False) //eliminar la rejilla vertical
OB SET(vSettings;Graph background color;"#573E82") //definir un color de fondo
OB SET(vSettings;Graph background opacity;40)
ARRAY TEXT($aTcols;2) //definir los colores para el gráfico
$aTcols{1}:="#B5CF32"
$aTcols{2}:="#D43A26"
OB SET ARRAY(vSettings;Graph colors;$aTcols)
GRAPH(vGraph;vSettings;X;A;B) //Dibujar el gráfico
```


80
72
64
56
48
40
32
24
16
8
0
Monday
Tuesday
Wednesday
Thursday
Friday
Mary
John

GRAPH SETTINGS

GRAPH SETTINGS (graf ; xmin ; xmax ; ymin ; ymax ; xprop ; xgrid ; ygrid ; titulo {; titulo2 ; ... ; tituloN})

Parámetro	Tipo		Descripción
graf	Variable imagen	➡	Área del gráfico o variable imagen
xmin	Entero largo, Fecha, Hora	➡	Valor mínimo del eje de las x para gráfico proporcional (líneas o puntos solamente)
xmax	Entero largo, Fecha, Hora	➡	Valor máximo del eje de las x para gráfico proporcional (líneas o puntos solamente)
ymin	Entero largo	➡	Valor mínimo del eje y
ymax	Entero largo	➡	Valor máximo del eje y
xprop	Booleano	➡	TRUE para eje x proporcional; FALSE para eje x normal (líneas o puntos solamente)
xgrid	Booleano	➡	TRUE para rejilla del eje x; FALSE para no rejilla en el eje x (sólo si xprop es TRUE)
ygrid	Booleano	➡	TRUE para rejilla del eje y; FALSE para no rejilla el eje y
titulo	Cadena	➡	Título(s) para las leyenda(s) del gráfico

Descripción

GRAPH SETTINGS *permite cambiar los parámetros de los gráficos mostrados en un formulario. El gráfico debe haber sido definido con el comando* ***GRAPH****.* ***GRAPH SETTINGS*** *no tiene efecto en un gráfico de tipo sectores. Este comando debe llamarse obligatoriamente en el mismo proceso que el formulario.*

Nota: *no debe llamar este comando si el gráfico se generó utilizando el comando* ***GRAPH*** *con el parámetro grafParams de tipo Objeto. Consulte la descripción del comando* ***GRAPH*** *para más información.*

Los parámetros xmin, xmax, ymin, y ymax fijan los valores mínimos y máximos para sus respectivos ejes del gráfico. Si el valor de un par de estos parámetros es un valor nulo, (0, ?00:00:00?, ó !00/00/00!, dependiendo del tipo de dato), se utilizarán los valores del gráfico por defecto. Los parámetros xmin y xmax sólo se tienen en cuenta para gráficos proporcionales (xprop es ***True****).*

El parámetro xprop activa el trazado proporcional para gráficos de líneas (tipo 4) y gráficos de puntos (tipo 6). Cuando este parámetro es TRUE, cada punto será trazado sobre el eje x de acuerdo al valore del punto, y luego sólo si los valores son numéricos, hora o fecha.

Nota: *con 4D Server 64 bits OS X, el parámetro xprop también se tiene en cuenta para los gráficos de tipo 5.*

Los parámetros xgrid y ygrid muestran u ocultan las líneas de rejilla. Una rejilla para el eje x será mostrada sólo si se trata de un gráfico de puntos o de líneas proporcional.

El(los) parámetro(s) titulo especifican los títulos de las leyendas.

Ejemplo

Ver el ejemplo del comando ***GRAPH****.*

_o_GRAPH TABLE

_o_GRAPH TABLE
Este comando no requiere parámetros

Descripción

***Comando eliminado**: Este comando ya no se soporta a partir de 4D v14.*

Grupos y usuarios

- *BLOB TO USERS*
- *CHANGE CURRENT USER*
- *CHANGE PASSWORD*
- *Current user*
- *DELETE USER*
- *EDIT ACCESS*
- *Get default user*
- *GET GROUP LIST*
- *GET GROUP PROPERTIES*
- *Get plugin access*
- *GET USER LIST*
- *GET USER PROPERTIES*
- *Is user deleted*
- *Set group properties*
- *SET PLUGIN ACCESS*
- *Set user properties*
- *User in group*
- *USERS TO BLOB*
- *Validate password*

BLOB TO USERS

BLOB TO USERS (usuarios)

Parámetro	Tipo		Descripción
usuarios	BLOB	➡	BLOB (encriptado) contiene las cuentas de usuarios creadas y guardadas por el Administrador

Descripción

El comando BLOB TO USERS remplaza las cuentas usuarios y los grupos creados por el Administrador en la base de datos con las que se encuentran en el BLOB usuarios. El BLOB usuarios está encriptado y debe haber sido creado utilizando el comando ***USERS TO BLOB****.*

Sólo el Administrador o el Diseñador, pueden ejecutar este comando. Si otro usuario intenta ejecutarlo, el comando no hace nada y se genera un error de privilegio (-9949).

Este comando hace que se remplacen todas las cuentas y grupos existentes creados por el Administrador de la base. Si el BLOB usuarios contiene datos válidos, el comando realiza las siguientes operaciones:

- *todos los usuarios y grupos definidos en la base cuyos números de referencia son negativos (grupos y usuarios creados por el administrador) se eliminan de la estructura,*
- *todos los usuarios y grupos que se encuentran en el BLOB usuarios se añaden a la estructura.*

Nota de compatibilidad: *los archivos de usuarios y grupos (extensión .4UG) creados por el comando de menú* ***Guardar Grupos...*** *en versiones de 4D anteriores pueden cargarse en 4D utilizando la siguiente secuencia (las versiones 4D muy antiguas pueden requerir el uso de versiones intermedias):*

```
DOCUMENT TO BLOB(mydoc;blob)
BLOB TO USERS(blob)
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0.

CHANGE CURRENT USER

CHANGE CURRENT USER {(usuario ; contraseña)}

Parámetro	Tipo		Descripción
usuario	Cadena, Entero largo	➡	Nombre o número de referencia único
contraseña	Cadena	➡	Contraseña (no encriptada)

Descripción

CHANGE CURRENT USER permite cambiar la identidad del usuario actual en la base, sin tener que salir. El usuario puede cambiar su identidad utilizando la caja de diálogo de identificación del usuario de la base (cuando el comando se llama sin parámetros) o directamente vía el comando. Cuando un usuario cambia su identidad, el usuario abandona sus privilegios de acceso anteriores para beneficiarse de los del usuario elegido.

Si el comando **CHANGE CURRENT USER** se ejecuta sin parámetros, se muestra la caja de diálogo de identificación del usuario de la base. El usuario debe entonces introducir o seleccionar un nombre y contraseña válidos para entrar a la base. El contenido de la caja de diálogo de conexión depende de las opciones definidas en la página **Seguridad** de las Preferencias de la base.

Nota: el sistema de control de acceso debe estar activado para poder utilizar este comando, es decir, una contraseña debe haber sido asignada al Diseñador. De lo contrario, **CHANGE CURRENT USER** no tiene ningún efecto y no se muestra la ventana estándar para cambiar de usuarios.

Igualmente puede pasar los parámetros opcionales usuario y contraseña para especificar por programación la nueva cuenta a utilizar.

Pase en el parámetro usuario el nombre o el número de referencia única (refUsuario) de la cuenta a utilizar. Los nombres y los números de usuario se pueden obtener utilizando el comando **GET USER LIST**.

Número de referencia del usuario	Descripción del usuario
1	Diseñador
2	Administrador
3 a 15000	Usuario creado por el Diseñador (el usuario No. 3 es el primer usuario creado por el Diseñador, usuario No. 4 es el segundo, etc.).
-11 a -15010	Usuario creado por el Administrador (usuario No. -11 es el primer usuario creado por el Administrador, usuario No. -12 es el segundo, etc.).

Si la cuenta de usuario designada no existe o fue borrada, se genera el error -9979. Puede interceptar este error con el método de gestión de errores instalado por el comando **ON ERR CALL**. De lo contrario, puede llamar la función **Is user deleted** para probar la cuenta de usuario antes de llamar este comando.

Pase en el parámetro contraseña la contraseña no encriptada de la cuenta de usuario. Si la contraseña no corresponde con el usuario, el comando devolverá el mensaje de error -9978 y no hará nada.

El comando es temporizado con el fin de evitar ataques de fuerza bruta, en otras palabras, intentos de múltiples combinaciones de nombres de usuario/contraseña. Como resultado, después de la cuarta llamada a este comando, no se ejecuta por un periodo de 10 segundos. Esta temporización es global a la estación de trabajo.

Ofrecer una caja de diálogo de gestión de acceso personalizada

El comando **CHANGE CURRENT USER** permite establecer cajas de diálogo personalizadas para introducir el nombre y contraseña (con reglas de entrada y de vencimiento) que tengan las mismas ventajas del sistema de control de accesos de 4D.

El principio es el siguiente:

1. La entrada en la base se efectúa directamente en modo "Usuario por defecto", sin caja de diálogo.

2. En el **Método de base On Startup,** el desarrollador provoca la visualización de una caja de diálogo personalizada de entrada del nombre de usuario y contraseña. Todos los tipos de procesos se pueden ver en la caja de diálogo:

- Es posible mostrar la lista de usuarios de la base, como en la caja de diálogo de acceso estándar de 4D, utilizando el comando **GET USER LIST**.

- El campo de entrada de la contraseña puede contener varios controles con el fin de verificar la validez de los caracteres introducidos (mínimo número de caracteres, unicidad, etc.).

- Para que los caracteres de contraseñas se introduzcan de manera que estén enmascarados en pantalla, puede utilizar el comando **FILTER KEYSTROKE** con la fuente especial %password.

- Las reglas de vencimiento pueden aplicarse en el momento en que la caja de diálogo se valida: fecha de vencimiento, cambio forzado a la conexión inicial, bloqueo de cuenta después de varias entradas incorrectas, memorización de contraseñas ya utilizadas, etc.

3. Cuando se valida la entrada, la información requerida (nombre de usuario y contraseña) se pasan al comando **CHANGE CURRENT USER** para abrir la base con los privilegios de la cuenta del usuario.

Ejemplo

El siguiente ejemplo muestra la caja de diálogo de identificación del usuario:

```
CHANGE CURRENT USER
```

CHANGE PASSWORD

CHANGE PASSWORD (contraseña)

Parámetro	Tipo		Descripción
contraseña	Cadena	➡	Nueva contraseña

Descripción

CHANGE PASSWORD cambia la contraseña del usuario actual. Este comando reemplaza la contraseña actual con la nueva contraseña que se pasa en contraseña.
Advertencia: las contraseñas diferencian los caracteres en mayúsculas y minúsculas.

Ejemplo

El siguiente ejemplo permite al usuario cambiar su contraseña.

```
CHANGE CURRENT USER  ` Mostrar la caja de diálogo de contraseñas
If(OK=1)
   $pw1:=Request("Introduzca la nueva contraseña para "+Current user)
 ` La contraseña debe tener al menos cinco caracteres
   If(((OK=1)&($pw1#""))&(Length($pw1)>5))
 ` Asegúrese de que la contraseña haya sido introducida correctamente
      $pw2:=Request("Introduzca de nuevo la contraseña")
      If((OK=1)&($pw1=$pw2))
         CHANGE PASSWORD($pw2)  ` Cambiar la contraseña
      End if
   End if
End if
```

Current user

Current user -> Resultado

Parámetro	***Tipo***	***Descripción***
Resultado	*Cadena*	*Nombre del usuario actual*

Descripción

Current user *devuelve el nombre del usuario actual.*

Ejemplo

Ver el ejemplo del comando ***User in group****.*

DELETE USER

DELETE USER (refUsuario)

Parámetro	Tipo		Descripción
refUsuario	Entero largo	➡	Número de identificación del usuario a borrar

Descripción

*El comando **DELETE USER** borra el usuario cuyo número se pasa en refUsuario. Debe pasar un número válido de usuario devuelto por el comando **GET USER LIST**.*

*Si la cuenta de usuario no existe o ha sido borrada, se genera el error -9979. Puede interceptar este error con un método de gestión de errores instalado por el comando **ON ERR CALL**.*

Sólo el Diseñador y el Administrador pueden borrar usuarios. El Administrador no puede borrar un usuario creado por el Diseñador.

*Los usuarios borrados no aparecerán más en el editor de usuarios cuando llame a **CHANGE ACCESS** ni en modo Diseño. Note que los números de usuarios borrados pueden reasignarse al crear nuevas cuentas.*

Gestión de errores

*Si no tiene los privilegios de acceso para llamar **DELETE USER** o si otro proceso ya accedió al sistema de contraseñas, se genera un error de privilegios de acceso. Puede interceptar este error con un método de gestión de errores instalado por el comando **ON ERR CALL**.*

EDIT ACCESS

EDIT ACCESS

Este comando no requiere parámetros

Descripción

EDIT ACCESS *permite modificar el sistema de contraseñas. Cuando se ejecuta este comando, se muestra la ventana de la caja de herramientas que contiene las páginas Usuarios y Grupos.*

Nota: *este comando abre una ventana modal. Por lo tanto, no de llamarlo desde otra ventana modal; de lo contrario no se abrirá la ventana y el comando no hará nada.*

Los grupos pueden ser modificados por el Diseñador, el Administrador y por los propietarios de grupo. El Diseñador y el Administrador pueden editar cualquier grupo. Los propietarios de grupo no pueden modificar sus propios grupos. Los usuarios pueden ser añadidos y retirados de grupos. El comando no tiene efecto si ningún grupo está definido.

El Diseñador y el Administrador pueden añadir nuevos usuarios, como también asignar usuarios a grupos.

Ejemplo

El siguiente ejemplo muestra al usuario la ventana de gestión de usuarios y grupos:

```
EDIT ACCESS
```

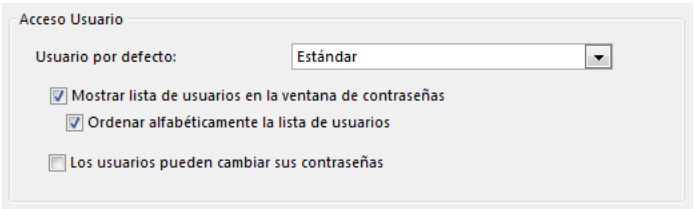
Get default user

Get default user -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Número de referencia único del usuario

Descripción

*El comando **Get default user** devuelve el número de referencia único del usuario designado como "Usuario por defecto" en la caja de diálogo de Preferencias de la base:*



Los siguientes números de referencia pueden ser utilizados por los usuarios:

Número de referencia	Descripción del usuario
1	Diseñador
2	Administrador
3 a 15000	Usuario creado por el Diseñador (usuario #3 es el primer usuario creado por el Diseñador, usuario #4 es el segundo, etc.).
-11 a -15010	Usuario creado por el Administrador (usuario #-11 es el primer usuario creado por el Administrador, usuario #-12 es el segundo, etc.).

Si ningún usuario por defecto está definido, el comando devuelve 0.

GET GROUP LIST

GET GROUP LIST (nomGrupos ; numGrupos)

Parámetro	Tipo		Descripción
nomGrupos	Array cadena	⇐	Nombres de los grupos tal como aparecen en el editor de contraseñas
numGrupos	Array entero largo	⇐	Números de referencia únicos para cada grupo

Descripción

GET GROUP LIST *llena los arrays nomGrupos y numGrupos con los nombres y los números de referencia únicos de los grupos tal como aparecen en el editor de contraseñas.*

El array numGrupos, sincronizado con el array nomGrupos, se llena con los números de referencia únicos de los grupos. Estos números pueden tener los siguientes rangos:

Número de referencia del grupo	Descripción del grupo
15001 a 32767	*Grupo creado por el Diseñador o por el propietario del grupo (grupo #15001 es el primer grupo creado por el Diseñador, grupo #15002 el segundo, etc.).*
-15001 a -32768	*Grupo creado por el Administrador o el propietario del grupo (grupo #-15001 es el primer grupo creado por el Administrador, grupo #-15002 el segundo, etc.).*

Gestión de errores

Si no tiene privilegios de acceso para llamar al comando ***GET GROUP LIST*** *o si otro proceso abrió en el sistema de contraseñas, se genera un error de privilegios de acceso. Puede interceptar este error con un método de gestión de errores instalado por* ***ON ERR CALL****.*

GET GROUP PROPERTIES

GET GROUP PROPERTIES (refGrupo ; nombre ; propietario {; miembros})

Parámetro	Tipo		Descripción
refGrupo	Entero largo	➡	Número de referencia del grupo
nombre	Cadena	⬅	Nombre del grupo
propietario	Entero largo	⬅	Número de referencia del propietario del grupo
miembros	Array entero largo	⬅	Miembros del grupo

Descripción

GET GROUP PROPERTIES devuelve las propiedades del grupo cuyo número de referencia se pasa en refGrupo. Pase el número de referencia del grupo devuelto por el comando **GET GROUP LIST**. Los números de referencia de los grupos pueden tener los siguientes valores o rangos:

Número de referencia del grupo	Descripción del grupo
	(grupo #15001 es el primer grupo creado por el Diseñador, grupo #15002 el segundo, etc.).
-15001 a -32768	Grupo creado por el Administrador o el propietario del grupo (grupo #-15001 es el primer grupo creado por el Administrador, grupo #-15002 el segundo, etc.).

Si no pasa un número de referencia válido, **GET GROUP PROPERTIES** devuelve parámetros vacíos.

Después de la llamada del comando, recupera el nombre y el número de propietario del grupo en los parámetros nombre y propietario.

Si pasa el parámetro opcional miembros, este array contendrá los números de referencia únicos de los usuarios que pertenecen al grupo. Los números de referencia de los miembros del grupo son los siguientes:

Número de referencia del miembro	Descripción del miembro
1	Diseñador
2	Administrador
3 a 15000	Usuario creado por el Diseñador de la base (usuario #3 es el primer usuario creado por el Diseñador, usuario #4 es el segundo, etc.).
-11 a -15000	Usuario creado por el Administrador de la base (usuario #-11 es el primer usuario creado por el Administrador, usuario #-12 es el segundo, etc.).
15001 a 32767	Grupo creado por el Diseñador o el propietario del grupo (grupo #15001 es el primer grupo creado por el Diseñador, grupo #15002 el segundo, etc.).
-15001 a -32768	Grupo creado por el Administrador o propietario del grupo (grupo #-15001 es el primer grupo creado por el Administrador, grupo #-15002 el segundo, etc.).

Gestión de errores

Si no tiene privilegios de acceso para llamar al comando **GET GROUP PROPERTIES** o si otro proceso abrió en el sistema de contraseñas, se genera un error de privilegios de acceso. Puede interceptar este error con un método de gestión de errores instalado por **ON ERR CALL**.

Get plugin access

Get plugin access (plugIn) -> Resultado

Parámetro	Tipo		Descripción
plugIn	Entero largo	➡	Número del plug-in
Resultado	Cadena	⮌	Nombre del grupo asociado con el plug-in

Descripción

*El comando **Get plugin access** devuelve el nombre del grupo de usuarios autorizados a utilizar el plug-in cuyo número se pasa en el parámetro plugIn. Si ningún grupo está asociado al plug-in, el comando devuelve una cadena vacía ("").*

Pase en el parámetro plugIn el número del plug-in del que quiere conocer el grupo de usuarios asociado. Las licencias de los plug-ins incluyen las licencias web y SOAP de 4D Client. Puede pasar una de las siguientes constantes del tema "":

Constante	Tipo	Valor
4D Client SOAP license	Entero largo	808465465
4D Client Web license	Entero largo	808465209
4D for OCI license	Entero largo	808465208
4D ODBC Pro license	Entero largo	808464946
4D View license	Entero largo	808465207
4D Write license	Entero largo	808464697

GET USER LIST

GET USER LIST (nomsUsuario ; refUsuario)

Parámetro	Tipo		Descripción
nomsUsuario	Array cadena	⇐	Nombres de los usuarios tal como aparecen en el editor de contraseñas
refUsuario	Array entero largo	⇐	Números de referencia únicos para cada usuario

Descripción

GET USER LIST *llena los arrays nomsUsuario y refUsuario con los nombres y los números de referencia únicos de los usuarios tal como aparecen en la ventana de contraseñas.*

El array nomsUsuario se llena con los nombres de usuarios mostrados en la ventana de contraseñas, incluyendo los usuarios cuyas cuentas están desactivadas (los nombres de los usuarios mostrados en verde en la ventana de contraseñas).

Nota: *utilice el comando **Is user deleted** para detectar los usuarios borrados.*

El array refUsuario, sincronizado con nomsUsuario, se llena con los números de referencia únicos de los usuarios. Estos números pueden tener los siguientes valores o rangos:

Número de referencia del usuario	Descripción del usuario
1	Diseñador
2	Administrador
3 a 15000	Usuario creado por el Diseñador de la base (usuario #3 es el primer usuario creado por el Diseñador, usuario #4 es el segundo, etc.).
-11 a -15000	Usuario creado por el Administrador de la base (usuario #-11 es el primer usuario creado por el Administrador, usuario #-12 es el segundo, etc.).

Gestión de errores

*Si no tiene privilegios de acceso para llamar al comando **GET USER LIST** o si otro proceso abrió en el sistema de contraseñas, se genera un error de privilegios de acceso. Puede interceptar este error con un método de gestión de errores instalado por **ON ERR CALL**.*

GET USER PROPERTIES

GET USER PROPERTIES (refUsuario ; nombre ; inicio ; contraseña ; nbLogin ; ultimoLogin {; membrecias {; grupoPropietario}})

Parámetro	Tipo		Descripción
refUsuario	Entero largo	➡	Número de referencia único de usuario
nombre	Cadena	⬅	Nombre del usuario
inicio	Cadena	⬅	Nombre del método de inicio
contraseña	Cadena	⬅	Cadena vacía
nbLogin	Entero largo	⬅	Números de usos de la base
ultimoLogin	Fecha	⬅	Fecha de la última utilización de la base
membrecias	Array entero largo	⬅	Números de referencia de los grupos a los que el usuario pertenece
grupoPropietario	Entero largo	⬅	Número de referencia del grupo prioritario del usuario

Descripción

GET USER PROPERTIES *devuelve la información sobre el usuario cuyo número de referencia se pasa en el parámetro refUsuario. Debe pasar un número de referencia de usuario devuelto por el comando* ***GET USER LIST****.*

Si la cuenta de usuario no existe o ha sido borrada, se genera el error -9979. Puede interceptar este error con un método de gestión de errores instalado por ***ON ERR CALL****. Sino, puede llamar* ***Is user deleted*** *para probar la cuenta de usuario antes de llamar* ***GET USER PROPERTIES****.*

Los números de referencia para los usuarios pueden tener los siguientes valores o rangos:

Número de referencia de usuario	Descripción usuario
1	Diseñador
2	Administrador
3 a 15000	Usuario creado por el Diseñador (usuario #3 es el primer usuario creado por el Diseñador, usuario #4 el segundo, etc.).
-11 a -15000	Usuario creado por el Administrador (usuario #-11 es el primer usuario creado por el Administrador, usuario #-12 el segundo, etc.).

Después de la llamada, recupera el nombre, método de inicio, contraseña encriptada, número de usos y la fecha de la última utilización de la base en los parámetros nombre, inicio, contraseña, nbLogin y ultimoLogin.

Nota: *el parámetro contraseña es obsoleto (siempre devuelve una cadena vacía). Si desea verificar la contraseña de un usuario, utilice la función* ***Validate password****.*

Si pasa el parámetro opcional membresias, recupera los números de referencia únicos de los grupos a los cuales pertenece el usuario.

Si pasa el parámetro opcional grupoProp, obtiene el número de referencia del grupo "propietario" del usuario, es decir el grupo propietario por defecto de los objetos creados por este usuario.

Los números de referencia para los grupos pueden ser los siguientes:

Número de referencia del grupo	Descripción del grupo
15001 a 32767	Grupo creado por el Diseñador o por el propietario del grupo (el grupo #15001 es el primer grupo creado por el Diseñador, el grupo #15002 el segundo, etc.).
-15001 a -32768	Grupo creado por el Administrador o por el propietario del grupo (el grupo #-15001 es el primer grupo creado por el Administrador, el grupo #-15002 el segundo, etc.).

Gestión de errores

Si no tiene privilegios de acceso para llamar al comando ***GET USER PROPERTIES*** *o si otro proceso abrió en el sistema de contraseñas, se genera un error de privilegios de acceso. Puede interceptar este error con un método de gestión de errores instalado por* ***ON ERR CALL****.*

Is user deleted

Is user deleted (refUsuario) -> Resultado

Parámetro	Tipo		Descripción
refUsuario	Entero largo	⇒	Número de identificación del usuario
Resultado	Booleano	⇐	TRUE = La cuenta del usuario ha sido borrada o no existe FALSE = La cuenta del usuario está activa

Descripción

*El comando **Is user deleted** prueba la cuenta de usuario cuyo número de identificación único se pasa en refUsuario.*
*Si la cuenta de usuario no existe o ha sido borrada, **Is user deleted** devuelve TRUE. De lo contrario, devuelve FALSE.*

Gestión de errores

*Si no tiene privilegios de acceso para llamar al comando **Is user deleted** o si otro proceso abrió en el sistema de contraseñas, se genera un error de privilegios de acceso. Puede interceptar este error con un método de gestión de errores instalado por **ON ERR CALL**.*

Set group properties

Set group properties (refGrupo ; nombre ; propietario {; miembros}) -> Resultado

Parámetro	Tipo		Descripción
refGrupo	Entero largo	➡	Número de referencia único del grupo activo o -1 para añadir un grupo de Diseñador o -2 para añadir un grupo de Administrador
nombre	Cadena	➡	Nuevo nombre de grupo
propietario	Entero largo	➡	Número de referencia único del usuario o del propietario del nuevo grupo
miembros	Array entero largo	➡	Nuevos miembros del grupo
Resultado	Entero largo	⮌	Número de referencia único del nuevo grupo

Descripción

Set group properties *permite modificar y actualizar las propiedades de un grupo existente cuyo número de referencia único se pasa en refGrupo, o para añadir un nuevo grupo afiliado al Diseñador o al Administrador.*

Si modifica las propiedades de un grupo existente, debe pasar un número de referencia válido devuelto por el comando ***GET GROUP LIST****. Los números de referencia de grupo son los siguientes:*

Número de referencia del grupo	Descripción del grupo
15001 a 32767	Grupo creado por el Diseñador o el propietario del grupo (el grupo #15001 es el primer grupo creado por el Diseñador, el grupo #15002 el segundo, etc.).
-15001 a -32768	Grupo creado por el Administrador o por el propietario del grupo (el grupo #-15001 el el primer grupo creado por el Administrador, el grupo #-15002 el segundo, etc.).

Para añadir un nuevo grupo afiliado con el Diseñador, pase -1 en ***RefGrupo****. Para añadir un nuevo grupo afiliado al Administrador, pase -2 en refGrupo. Después de la llamada, si el grupo se añadió con éxito, devuelve su número de referencia único en refGrupo.*

Si no pasa -1, -2 o un número de referencia de grupo válido, ***Set group properties*** *no hace nada y devuelve 0.*

Antes de llamar esta rutina, pase el nuevo nombre del grupo y el número de propietario del grupo en los parámetros nombre y propietario.Si no quiere cambiar las propiedades del grupo (a parte de los miembros, ver más adelante), primero llame ***GET GROUP PROPERTIES*** *y pase los parámetros que quiere dejar intactos.*

Si no pasa el parámetro opcional miembros, la lista actual de miembros del grupo permanece sin cambios. Si no pasa miembros mientras añade un grupo, el grupo no tendrá miembros.

Nota: *el propietario de un grupo no está definido automáticamente como miembro del grupo que posee. Es su decisión incluir al propietario del grupo en el grupo, utilizando el parámetro miembros.*

Si pasa el parámetro opcional miembros, modifica toda la lista de miembros para este grupo. Antes de llamar esta rutina, debe llenar el array miembros con los números de referencia únicos de los usuarios y grupos que el grupo tendrá como miembros. Los números de referencia de los miembros pueden tener los siguientes rangos:

Número de referencia del miembro	Descripción miembro
1	Diseñador
2	Administrador
3 a15000	Usuario creado por el Diseñador (el usuario #3 es el primer usuario creado por el Diseñador, el usuario #4 es el segundo, etc.).
-11 a -15000	Usuario creado por el Administrador (el usuario #-11 es el primer usuario creado por el Administrador, usuario #-12 es el segundo, etc.).
15001 a 32767	Grupo creado por el Diseñador o Propietario del grupo (el grupo #15001 es el primer grupo creado por el Diseñador, el grupo #15002 es el segundo, etc.).
-15001 a -32768	Grupo creado por el Administrador o Propietario del grupo (el grupo #-15001 es el primer grupo creado por el Administrador, el grupo #-15002 es el segundo, etc.).

Para eliminar todos los miembros de un grupo, pase un array vacío en el parámetro miembros.

Gestión de errores

Si no tiene privilegios de acceso para llamar al comando ***Set group properties*** *o si otro proceso abrió en el sistema de contraseñas, se genera un error de privilegios de acceso. Puede interceptar este error con un método de gestión de errores instalado por* ***ON ERR CALL****.*

SET PLUGIN ACCESS

SET PLUGIN ACCESS (plugIn ; grupo)

Parámetro	Tipo		Descripción
plugIn	Entero largo	➡	Número del plug-in
grupo	Cadena	➡	Nombre del grupo a asociar al plug-in

Descripción

*El comando **SET PLUGIN ACCESS** permite especificar por programación el grupo de usuarios autorizado a utilizar cada plug-in "serializado" instalado en la base. Al hacer esto, puede administrar la repartición de las licencias de los plug-ins.*

***Nota:** esta operación también puede efectuarse en modo Diseño en el editor de grupos.*

Pase en el parámetro plugIn el número del plug-in a asociar un grupo de usuarios. Las licencias de plug-ins incluyen las licencias web y SOAP de 4D Client. Puede pasar una de las siguientes constantes del tema "":

Constante	Tipo	Valor
4D Client SOAP license	Entero largo	808465465
4D Client Web license	Entero largo	808465209
4D for OCI license	Entero largo	808465208
4D ODBC Pro license	Entero largo	808464946
4D View license	Entero largo	808465207
4D Write license	Entero largo	808464697

Pase en el parámetro grupo el nombre del grupo cuyos usuarios están autorizados a utilizar el plug-in.

***Nota:** sólo un grupo a la vez puede utilizar un plug-in. Cuando este comando se ejecuta, si otro grupo tiene los derechos de acceso al plug-in, se pierde este privilegio.*

Set user properties

Set user properties (refUsuario ; nombre ; inicio ; contraseña ; nbLogin ; ultimoLogin {; membrecias {; grupoPropietario}}) -> Resultado

Parámetro	Tipo		Descripción
refUsuario	Entero largo	➡	Número de referencia único de cuenta del usuario o -1 para añadir un usuario afiliado al Diseñador o -2 para añadir un usuario afiliado al Administrador
nombre	Cadena	➡	Nuevo nombre de usuario
inicio	Cadena	➡	Nombre del nuevo método de inicio
contraseña	Cadena	➡	Nueva contraseña (encriptada) o * para no modificar la contraseña
nbLogin	Entero largo	➡	Nuevo número de usos de la base
ultimoLogin	Fecha	➡	Nueva fecha de la última utilización de la base
membrecias	Array entero largo	➡	Números de referencia de los grupos a los que pertenece el usuario
grupoPropietario	Entero largo	➡	Número de referencia del grupo propietario del usuario
Resultado	Entero largo	↩	Número de referencia único del nuevo usuario

Descripción

Set user properties permite modificar y actualizar las propiedades de una cuenta activa de usuario existente cuyo número de referencia se pasa en el parámetro refUsuario o para añadir un nuevo usuario afiliado al Diseñador o al Administrador.

Si cambia las propiedades de un usuario existente, debe pasar el número de referencia devuelto por el comando **GET USER LIST**.

Si la cuenta de usuario no existe o se ha borrado, se genera el error -9979. Puede interceptar este error con un método de gestión de errores instalado por **ON ERR CALL**. De lo contrario, puede llamar **Is user deleted** para probar la cuenta de usuario antes de llamar **Set user properties**.

Los números de referencia para los usuarios pueden ser los siguientes:

Número de referencia del usuario	Descripción usuario
1	Diseñador
2	Administrador
3 a 15000	Usuario creado por el Diseñador (el usuario #3 es el primer usuario creado por el Diseñador, el usuario #4 el segundo, etc.).
-11 a -15000	Usuario creado por el Administrador (el usuario #-11 es el primer usuario creado por el Administrador, el usuario #-12 el segundo, etc.).

Para añadir un nuevo usuario afiliado al Diseñador pase -1 en refUsuario. Para añadir un nuevo usuario afiliado al Administrador pase -2 en refUsuario.

Después de la llamada, si el usuario se añade o modifica con éxito, su número de referencia único es devuelto en refUsuario.

Si no pasa -1, -2 o un número de referencia de usuario válido, **Set user properties** no hace nada.

Antes de llamar este comando, pase el nuevo nombre, método de inicio, contraseña, número de usos y la fecha del último uso del usuario, en nombre, inicio, contraseña, nbLogin y ultimoLogin. Pase una contraseña no encriptada en el parámetro contraseña. 4D la encriptará por usted antes de guardarla en la cuenta de usuario.

Si el nuevo nombre de usuario pasado en nombre no es único (existe un usuario con el mismo nombre), el comando no hace nada y se devuelve el error -9979. Puede interceptar este error con un método de gestión de errores instalado por **ON ERR CALL**.

Si no quiere cambiar todas las propiedades del usuario (a parte de su grupo, ver a continuación), primero llame **GET USER PROPERTIES** y pase los valores devueltos para las propiedades que no quiere cambiar.

Si no quiere modificar la contraseña de una cuenta, pase el símbolo * en el parámetro contraseña. Esto le permite cambiar otras propiedades de la cuenta del usuario sin cambiar la contraseña de la cuenta.

Si no pasa el parámetro opcional membrecias, las membrecías actuales del usuario permanecen iguales. Si no pasa membrecias cuando añade un usuario, el usuario no formará parte de ningún grupo.

Si pasa el parámetro opcional membrecias, cambia todas las membrecías para el usuario. Antes de llamar este comando, debe llenar el array membrecias con los números de referencia únicos de los grupos cuyos usuarios harán parte.

Si pasa el parámetro opcional grupoPropietario, indica el número de referencia del grupo "propietario", del usuario, es decir el grupo propietario por defecto de los objetos creados por este usuario.

Los números de referencia para los grupos pueden ser los siguientes:

Número de referencia del grupo	Descripción del grupo
15001 a 32767	Grupo creado por el Diseñador o por el propietario del grupo (el grupo #15001 es el primer grupo creado por el Diseñador, el grupo #15002 es el segundo, etc.).
-15001 a -32768	Grupo creado por el Administrador o por el Propietario del grupo (grupo #-15001 es el primer grupo creado por el Administrador, el grupo #-15002 es el segundo, etc.).

Para anular todas las membrecías de un usuario, pase un array vacío en el parámetro membrecias.

Gestión de errores

*Si no tiene privilegios de acceso para llamar al comando **Set user properties** o si otro proceso abrió en el sistema de contraseñas, se genera un error de privilegios de acceso. Puede interceptar este error con un método de gestión de errores instalado por **ON ERR CALL**.*

User in group

User in group (usuario ; grupo) -> Resultado

Parámetro	Tipo		Descripción
usuario	Cadena	⇒	Nombre del usuario
grupo	Cadena	⇒	Nombre del grupo
Resultado	Booleano	⇐	TRUE = el usuario está en el grupo FALSE = el usuario no está en el grupo

Descripción

User in group *devuelve TRUE si usuario está en grupo.*

Ejemplo

El siguiente ejemplo busca facturas específicas. Si el usuario actual está en el grupo Administración, podrá acceder a los formularios que muestran información confidencial. Si el usuario no está en el grupo Administración, aparecen los formularios estándar:

```
QUERY([Facturas];[Facturas]Precio>100)
If(User in group(Current user;"Administración"))
   FORM SET OUTPUT([Facturas];"Confidencial_Salida")
   FORM SET INPUT([Facturas];"Confidencial_Entrada")
Else
   FORM SET OUTPUT([Facturas];"Salida_Estandar")
   FORM SET INPUT([Facturas];"Entrada_Estandar")
End if
MODIFY SELECTION([Facturas];*)
```

USERS TO BLOB

USERS TO BLOB (usuarios)

Parámetro	*Tipo*		*Descripción*
usuarios	*BLOB*	➡	*BLOB que debe contener los usuarios*
		⬅	*Cuentas de usuarios (encriptado)*

Descripción

*El comando **USERS TO BLOB** guarda en el BLOB usuarios la lista de todas las cuentas de usuarios y los grupos de la base creados por el Administrador.*

Sólo el Administrador y el Diseñador de la base pueden ejecutar este comando. Si otro usuario intenta ejecutarlo, el comando no hace nada y se genera un error de privilegio (-9949).

*El BLOB generado se encripta automáticamente y sólo puede ser leído utilizando el comando **BLOB TO USERS**. Puede almacenar este BLOB en un archivo en su disco duro o en un campo.*

Este comando es el equivalente al registro de los grupos y usuarios desde la ventana de gestión de los grupos de la Caja de herramientas. La única diferencia es que permite almacenar cuentas de usuarios en un campo BLOB y no únicamente en un archivo.

Este concepto permite conservar un backup de usuarios en la base e implementar un mecanismo de backup como también un sistema para cargar automáticamente a los usuarios en caso de una actualización de la estructura de la base (la información relacionada con las cuentas de usuario se son guardadas por 4D en el archivo de estructura de la base).

Validate password

Validate password (refUsuario ; contraseña {; digest}) -> Resultado

Parámetro	Tipo		Descripción
refUsuario	Entero largo, Cadena	➡	Número de referencia único
contraseña	Cadena	➡	Contraseña no encriptada
digest	Booleano	➡	Contraseña digest = True, Contraseña texto plano (por defecto) = False
Resultado	Booleano	⮌	True = contraseña correcta False = contraseña incorrecta

Descripción

Validate password *devuelve True si la cadena pasada en contraseña es la contraseña para la cuenta de usuario cuyo número de referencia se pasa en refUsuario.*

El parámetro opcional digest indica si el parámetro contraseña contiene una contraseña en texto plano o na contraseña en forma hash (modo digest):

- *si pasa **True**, indica que el parámetro contraseña contiene una contraseña en forma hash (modo digest),*
- *si pasa **False** u omite este parámetro, indica que contraseña contiene una contraseña en texto plano.*

*Este parámetro es particularmente útil cuando se utilizan métodos base de autenticación, en particular el **Método base On 4D Mobile Authentication**.*
El comando es temporizado con el fin de evitar ataques de fuerza bruta, en otras palabras, intentos de múltiples combinaciones de nombres de usuario/contraseña. Como resultado, después de la cuarta llamada a este comando, no se ejecuta por un periodo de 10 segundos. Esta temporización es global a la estación de trabajo.

Ejemplo 1

Este ejemplo verifica si la contraseña del usuario "Hardy" es "Laurel":

```
GET USER LIST(atNombreUsuario;alRefUsuario)
$vlElem:=Find in array(atNombreUsuario;"Hardy")
If($vlElem>0)
    If(Validate password(alRefUsuario{$vlElem};"Laurel"))
        ALERT("¡Sí!")
    Else
        ALERT("¡Error!")
    End if
Else
    ALERT("Nombre de usuario desconocido")
End if
```

Ejemplo 2

*En el **Método base On 4D Mobile Authentication**, usted puede probar una petición de conexión (utilizando los usuarios 4D de la base). Puede escribir:*

```
$0:=Validate password($1;$2;$3)
```

Herramientas

- *BASE64 DECODE*
- *BASE64 ENCODE*
- *Choose*
- *Generate digest*
- *Generate password hash*
- *Generate UUID*
- *GET ACTIVITY SNAPSHOT*
- *GET MACRO PARAMETER*
- *LAUNCH EXTERNAL PROCESS*
- *Load 4D View document*
- *OPEN URL*
- *PROCESS 4D TAGS*
- *SET ENVIRONMENT VARIABLE*
- *SET MACRO PARAMETER*
- *Verify password hash*

BASE64 DECODE

BASE64 DECODE ({textoEncode ;} BLOB)

Parámetro	Tipo		Descripción
textoEncode	Texto	➡	Texto que contiene un BLOB codificado en base64
BLOB	BLOB	➡	BLOB codificado en formato base64
		⬅	BLOB decodificado

Descripción

*El comando **BASE64 DECODE** permite decodificar el BLOB codificado en formato base64 pasado en el parámetro textoEncode o blob.*

Si pasa el parámetro textoEncode, el comando decodifica su contenido y lo devuelve en el parámetro blob. Debe contener un BLOB codificado en formato Base64. En este caso, el contenido inicial del parámetro blob es ignorado por el comando.
Si omite el parámetro textoEncode, el comando modifica directamente el BLOB pasado en el parámetro blob.

El comando no verifica el contenidos del parámetro textoEncode o blob. Debe verificar que los datos pasados estén efectivamente codificados en formato base64, de lo contrario el resultado será incorrecto.

Ejemplo

Este ejemplo le permite transferir una imagen vía un BLOB:

```
C_BLOB($sourceBlob)
C_PICTURE($mypicture)
$mypicture:=[people]photo
PICTURE TO BLOB($mypicture;$sourceBlob;".JPG")
C_TEXT($base64Text)
BASE64 ENCODE($sourceBlob;$base64Text) //Encoding of text
 // the binary is now available as character strings in $base64Text

C_TEXT($base64Text)
C_BLOB($targetBlob)
BASE64 DECODE($base64Text;$targetBlob) //Decoding of text
 // the binary encoded in base 64 is now available as a BLOB in $blobTarget
```

BASE64 ENCODE

BASE64 ENCODE (BLOB {; textoEncode})

Parámetro	*Tipo*		*Descripción*
BLOB	*BLOB*	➡	*BLOB para codificar en formato base64*
		⬅	*BLOB codificado en formato base64*
textoEncode	*Texto*	⬅	*Resultado del blob codificado en base64*

Descripción

*El comando **BASE64 ENCODE** codifica el BLOB pasado en el parámetro blob en formato base64.*

Si pasa el parámetro textoEncode, recibe los contenidos del blob como texto al final de la ejecución del comando. En este caso, el parámetro blob mismo no es modificado por el comando.
Si omite el parámetro textoEncode, el comando modifica directamente el BLOB pasado como parámetro.

La codificación base64 modifica los datos codificados sobre 8 bits de manera que no conserven más de 7 bits útiles. Esta codificación es necesaria, por ejemplo, para la manipulación de BLOBs utilizando XML.

Choose

Choose (criterio ; valor {; valor2 ; ... ; valorN}) -> Resultado

Parámetro	Tipo		Descripción
criterio	Booleano, Entero	➡	Valor a probar
valor	Expresión	➡	Valores posibles
Resultado	Expresión	⮌	Valor de criterio

Descripción

*El comando **Choose** devuelve uno de los valores pasados en los parámetros valor1, valor2, etc. en función del valor del parámetro criterio.*

Puede pasar un parámetro criterio de tipo booleano o numérico:

- *Si criterio es un booleano, **Choose** devuelve valor1 si el booleano es igual a True y valor2 si el booleano es igual a False. En este caso, el comando espera exactamente tres parámetros: criterio, valor1 y valor2.*
- *Si criterio es un entero, **Choose** devuelve el valor cuya posición corresponde a criterio. Atención, la numeración de los valores comienza en 0 (la posición de valor1 es 0). En este caso, el comando espera al menos dos parámetros: criterio y valor1.*

El comando acepta todo los tipos de datos para el/los parámetro(s) valor, excepto imágenes, punteros, BLOBS y arrays. Sin embargo, debe asegurarse de que todos los valores pasados sean del mismo tipo, 4D no efectuará ninguna verificación en este punto.

*Si ningún valor corresponde a criterio, **Choose** devuelve un valor "nulo" con respecto al tipo del parámetro valor (por ejemplo, 0 para el tipo numérico, "" para el tipo cadena, etc.).*

Este comando permite generar código conciso que reemplaza las pruebas de tipo "Case of" que toman varias líneas (ver ejemplo 2). También es muy útil en los lugares donde pueden ejecutarse fórmulas: editor de búsquedas, aplicar una fórmula, editor de informes rápidos, columna calculada de listbox, etc.

Ejemplo 1

Este es un ejemplo del uso típico de este comando con un criterio de tipo booleano:

```
vTitulo:=Choose([Persona]Masculino;"Sr";"Sra")
```

Este código es estrictamente equivalente a:

```
If([Persona]Masculino)
   vTitulo:="Sr"
Else
   vTitulo:="Sra"
End if
```

Ejemplo 2

Este es un ejemplo del uso típico de este comando con un criterio de tipo numérico:

```
vEstado:=Choose([Persona]Estado;"Soltero";"Casado";"Viudo";"Separado")
```

Este código es estrictamente equivalente a:

```
Case of
   :([Persona]Estado=0)
      vEstado:="Soltero"
   :([Persona]Estado=1)
      vEstado:="Casado"
   :([Persona]Estado=2)
      vEstado:="Viudo"
   :([Persona]Estado=3)
      vEstado:="Separado"
End case
```

Generate digest

Generate digest (param ; algoritmo) -> Resultado

Parámetro	Tipo		Descripción
param	BLOB, Variable texto	➡	Blob o texto para el cual obtener un extracto
algoritmo	Entero largo	➡	Algoritmo utilizado para devolver la llave: 0 = Digest MD5, 1 = Digest SHA1
Resultado	Texto	⮌	Valor del extracto

Descripción

*El comando **Generate digest** devuelve el extracto de un BLOB o de un texto después de la aplicación de un algoritmo de encripción.*

*Pase un campo o una variable Texto o BLOB en el parámetro param. La función **Generate digest** devuelve la llave digest como una cadena.*

*En el parámetro algoritmo, pase un valor designando la función hash a utilizar. Utilice una de las siguientes constantes, ubicadas en el tema **Tipo Digest**:*

Constante	Tipo	Valor	Comentario
4D digest	Entero largo	2	Utilizar el algoritmo interno de 4D
MD5 digest	Entero largo	0	Utilizar el algoritmo MD5
SHA1 digest	Entero largo	1	Utilizar el algoritmo SHA-1
SHA256 digest	Entero largo	3	(Familia SHA-2) SHA-256 es una serie de 256 bits devueltos como una cadena de 64 caracteres hexadecimales.
SHA512 digest	Entero largo	4	(Familia SHA-2) SHA-512 es una serie de 512 bits devueltos como una cadena de 128 caracteres hexadecimales.

***Nota:** no se recomienda utilizar algoritmos MD5 y SHA para manejar contraseñas; si necesita verificar contraseñas, se recomienda utilizar los comandos **Generate password hash** y **Verify password hash** .*

El valor devuelto por el mismo objeto es el mismo en todas las plataformas (Mac/Windows, 32 o 64 bits). El cálculo se efectúa a partir de la representación en UTF-8 del texto pasado en parámetro.

***Nota:** si utiliza el comando con un texto/BLOB vacío, no devolverá void sino un valor cadena (por ejemplo "d41d8cd98f00b204e9800998ecf8427e" para MD5).*

Ejemplo 1

Este ejemplo compara dos documentos utilizando el algoritmo MD5:

```
C_PICTURE($vPict1;$vPict2)
C_BLOB($FirstBlob;$SecondBlob)
READ PICTURE FILE("c:\\myPhotos\\photo1.png")
If(OK=1)
   READ PICTURE FILE("c:\\myPhotos\\photo2.png")
   If(OK=1)
      PICTURE TO BLOB($vPict1;$FirstBlob;".png")
      PICTURE TO BLOB($vPict2;$SecondBlob;".png")

      $MD5_1:=Generate digest($FirstBlob;MD5 digest)
      $MD5_2:=Generate digest($SecondBlob;MD5 digest)

      If($MD5_1#$MD5_2)
         ALERT("Estas dos imágenes son diferentes.")
      Else
         ALERT("Estas dos imágenes son idénticas.")
      End if
   End if
End if
```

Ejemplo 2

Estos ejemplos ilustran cómo recuperar el extracto de un texto:

```
$key1:=Generate digest("The quick brown fox jumps over the lazy dog.";MD5 digest)
 // $key1 is "e4d909c290d0fb1ca068ffaddf22cbd0"
```

```
$key2:=Generate digest("The quick brown fox jumps over the lazy dog.";SHA1 digest)
 // $key2 is "408d94384216f890ff7a0c3528e8bed1e0b01621"
```

Ejemplo 3

Este ejemplo sólo acepta el usuario "admin" con la contraseña "123" que no corresponde a un usuario 4D:

```
 //On REST Authentication database method
C_TEXT($1;$2)
C_BOOLEAN($0;$3)
 //$1: usuario
 //$2: contraseña
 //$3: modo digest
If($1="admin")
    If($3)
        $0:=($2=Generate digest("123";4D digest))
    Else
        $0:=($2="123")
    End if
Else
    $0:=False
End if
```

Generate password hash

Generate password hash (contrasena {; opciones}) -> Resultado

Parámetro	Tipo		Descripción
contrasena	Cadena	➡	La contraseña del usuario. Sólo se utilizan los primeros 72 caracteres.
opciones	Objeto	➡	Un objeto que contiene opciones.
Resultado	Cadena	⮌	Devuelve la contraseña hash.

Descripción

La función ***Generate password hash*** *devuelve un hash de contrasena seguro generado por un algoritmo de hash criptográfico.*

Pase un valor de cadena en el parámetro contrasena. ***Generate password hash*** *devuelve una cadena de hash para la contraseña. Múltiples pases de la misma contraseña darán lugar a cadenas hash diferentes.*

En el objeto opciones, pase las propiedades que se utilizarán al generar el hash de la contraseña. Los valores disponibles se muestran en la siguiente tabla:

Propiedad	Tipo de valor	Descripción	Valor por defecto
algorithm	cadena	algoritmo que se utilizará. Actualmente sólo se admite "bcrypt" (sensible a mayúsculas y minúsculas).	bcrypt
cost	numérico	velocidad que se utilizará. Los valores admitidos para bcrypt están entre 4 y 31.	10

Nota*: si un valor en el objeto de opciones no es válido, se devolverá un mensaje de error y una cadena vacía.*

Gestión de errores

Se pueden devolver los siguientes errores. Puede revisar un error con los comandos ***GET LAST ERROR STACK*** *y* ***ON ERR CALL****.*

Número	Mensaje
850	Password-hash: Algoritmo no soportado.
852	Password-hash: No disponible bcrypt costo parámetro, ofrece un valor entre 4 y 31.

Sobre bcrypt

bcrypt es una función de hashing de contraseñas basada en el cifrado Blowfish. Además de incorporar una sal para proteger contra los ataques tabla arco iris, es una función adaptativa en la que el recuento de la iteración puede aumentarse para hacerla más lenta, por lo que sigue siendo resistente a los ataques de fuerza bruta incluso con el aumento del poder computacional, porque toma demasiado tiempo y es costoso.

Ejemplo

Este ejemplo genera un hash de contraseña utilizando bcrypt con un factor de costo 4.

```
C_TEXT($password)
C_TEXT($hash)
C_OBJECT($options)

$options:=New object("algorithm";"bcrypt";"cost";4)
$password:=Request("Please enter your password")

$hash:=Generate password hash($password;$options)
[Users]hash:=$hash
SAVE RECORD([Users])
```

Nota*: múltiples pasadas de la misma contraseña darán lugar a cadenas hash diferentes. Este es un comportamiento estándar para algoritmos como bcrypt, ya que la mejor práctica es crear una nueva sal aleatoria para cada hash. Consulte la descripción* ***Verify password hash*** *para ver un ejemplo de cómo comprobar las contraseñas.*

Generate UUID

Generate UUID -> Resultado

Parámetro	Tipo		Descripción
Resultado	Cadena		Nuevo UUID en forma de texto no canónico (32 caracteres)

Descripción

Generate UUID *devuelve un nuevo identificador UUID de 32 caracteres en forma no canónica.*

Un UUID es un número de 16 bytes (128 bits). Contiene 32 caracteres hexadecimales. Puede expresarse en forma no canónica (series de 32 letras [A-F, a-f] y/o números [0-9], por ejemplo 550e8400e29b41d4a716446655440000) o en forma canónica (grupos de 8,4,4,4,12, por ejemplo 550e8400-e29b-41d4-a716-446655440000).

En 4D, los números UUID pueden guardarse en campos. Para mayor información, consulte la sección en el manual de Diseño.

Ejemplo

Generación de un UUID en una variable:

```
C_TEXT(MyUUID)
MyUUID:=Generate UUID
```

GET ACTIVITY SNAPSHOT

GET ACTIVITY SNAPSHOT (arrActividades | arrUUID ; arrInicio ; arrDuracion ; arrInfo {; arrDetails}{; *})

Parámetro	Tipo		Descripción
arrActividades | arrUUID	Array objeto, Array texto	⬅	Descripción completa de operaciones (array objeto) o UUIDs de las operaciones (array texto)
arrInicio	Array texto	⬅	Horas de inicio de las operaciones
arrDuracion	Array entero largo	⬅	Duración de las operaciones en segundos
arrInfo	Array texto	⬅	Descripción
arrDetails	Array objeto	⬅	Detalles del contexto y sub operaciones (si las hay)
*	Operador	➡	Si se pasa = Traer actividad del servidor

Descripción

*El comando **GET ACTIVITY SNAPSHOT** retorna un array o varios que describen las operaciones en progreso sobre los datos 4D. Estas operaciones usualmente muestran una ventana de progreso.*

Este comando se usa para traer una imagen de las x operaciones que más consumen tiempo tiempo y/o que corren más frecuentemente, tales como escritura de caché o ejecución de fórmulas.

***Nota:** La información devuelta por el comando **GET ACTIVITY SNAPSHOT** es la misma mostrada en la página "Monitor en tiempo real" (RTM) de la ventana de administración de 4D Server (vea el Manual de 4D Server).*

*Por defecto, **GET ACTIVITY SNAPSHOT** procesa las operaciones realizadas a nivel local (con 4D monopuesto, 4D Server o 4D en modo remoto). Sin embargo, con 4D en modo remoto, también puede obtener una instantánea de las operaciones realizadas en el servidor: sólo tiene que pasar el asterisco (*) como último parámetro. En este caso, el servidor de datos se recupera localmente.*
*El parámetro * se ignora cuando el comando se ejecuta en 4D Server o 4D monopuesto.*

*El comando **GET ACTIVITY SNAPSHOT** acepta dos sintaxis:*

- *sintaxis usando solamente un array de objetos.*
- *sintaxis utilizando varios arrays.*

Primera sintaxis: GET ACTIVITY SNAPSHOT ({* ;} arrActivities)

Con esta sintaxis, todas las operaciones se devuelven en un formulario estructurado en el array objetos 4D (arrActividades). Cada elemento del array es un objeto construido de la siguiente manera:

```
[
    {
        "message":"xxx",
        "maxValue":12321,
        "currentValue":63212,
        "interruptible:0,
        "remote":0,
        "uuid":"deadbeef",
        "taskId":xxx,
        "startTime":"2014-03-20 13:37:00:123",
        "duration":92132,
        "dbContextInfo":{
            "task_id": xxx,
            "user_name": Jean,
            "host_name": HAL,
            "task_name": "CreateIndexLocal",
            "client_uid": "DE4DB33F33F"
            "user4d_id ": 1,
            "client_version ": 123456
        },
        "dbOperationDetails":{
            table: "myTable"
            field: "Field_1"
        },
        "subOperations":[
            {"message":"xxx",
            ...}
            ]
    },
    {...}
]
```

Esta es un descripción de cada propiedad devuelta:

- *message (texto): etiqueta de la operación*

- *maxValue (número): número de iteraciones definidas para la operación (-1 si la operación no es iterativa)*
- *currentValue (número): iteración actual*
- *interruptible (número): la operación puede ser interrumpida por el usuario (0=true, 1=false)*
- *remote (número): operación por pares entre cliente y servidor (0=true, 1=false)*
- *uuid (text): identificador UUID de la operación*
- *taskId (número): identificador interno del proceso en el origen de la operación*
- *startTime (texto): la hora de inicio de la operación en formato "aaaa:mm:dd hh:mm:ss:mls"*
- *duration (número): duración de la operación en milisegundos*
- *dbContextInfo (objeto): información relativa a las operaciones manejadas por el motor de la base de datos. Contiene las siguientes propiedades:*
  - *host_name (cadena): nombre del host que lanzó la operación*
  - *user_name (cadena): nombre del usuario 4D cuya sesión lanzó la operación*
  - *task_name (cadena): nombre del proceso que lanzó la operación*
  - *task_id (num): número del ID del proceso que lanzó la operación*
  - *client_uid (cadena): opcional, uuid del cliente que lanzó la operación*
  - *is_remote_context (booleano, 0 o 1): opcional, indica si la operación de la base fue lanzada por un cliente (valor 1) o por el servidor por medio del procedimiento almacenado (valor 0)*
  - *user4d_id (num): número del ID del usuario 4D actual del lado del cliente*
  - *client_version (cadena): cuatro dígitos representan la versión del motor 4D de la aplicación, como los devolvió el comando **Application version**.*

  ***Nota:** client_uid and is_remote_context sólo está disponible en modo cliente/servidor. client_uid sólo se devuelve si la operación de la base de datos se inició en un equipo cliente.*
- *dbOperationDetails (objeto): propiedad devuelta únicamente si la operación llama al motor de base de datos (este es el caso, por ejemplo, para búsqueda y ordenaciones). Este es un objeto que contiene información específica relacionada con la operación en sí. Las propiedades disponibles dependen del tipo de la operación de base de datos realizada. Más específicamente, estas propiedades incluyen:*
  - *table (cadena): nombre de la tabla implicado en la operación*
  - *field (cadena): nombre del campo implicado en la operación*
  - *queryPlan (cadena): plan de búsqueda definido para la operación*
  - *...*
- *subOperations (array): array de objetos que contienen sub-operaciones de la operación actual (si existe). La estructura de cada sub-elemento es idéntica a la del objeto principal. Si la operación actual no tiene sub-operaciones, entonces subOperations está vacío.*

Segunda sintaxis: GET ACTIVITY SNAPSHOT ({ ;} arrUUID ; arrStart ; arrDuration ; arrInfo {;arrSubOp})*

Con esta sintaxis, todas las operaciones se devuelven en varios arrays sincronizados (cada operación provoca que un elemento se añada a todos los arrays). Los siguientes arrays se devuelven:

- *arrUUID: contiene los identificadores UUID de cada operación (corresponde a la propiedad uuid del objeto arrActividades en la sintaxis anterior).*
- *arrInicio: contiene las horas de inicio de cada operación (corresponde a la propiedad startTime del objeto arrActividades).*
- *arrDuracion: contiene las duraciones de cada operación en milisegundos (corresponde a la propiedad duration del objeto arrActividades).*
- *arrInfo: contiene las etiquetas que describen cada operación (corresponde a la propiedad message del objeto arrActividades).*
- *arrDetalles (opcional): cada elemento de este array es un objeto que contiene las siguientes propiedades:*
  - *"dbContextInfo" (objeto): ver arriba*
  - *"dbOperationDetails" (objeto): ver arriba*
  - *"subOperaciones". El valor de esta propiedad es un array objeto que contiene todas las sub-operaciones de la operación actual. Si la operación actual no tiene sub-operaciones, el valor de la propiedad suboperaciones es un array vacío. (corresponde a la propiedad subOperations del objeto arrActividades).*

Ejemplo

Este método, ejecutado en un proceso separado en 4D o 4D Server, ofrece una instantánea de las operaciones que están en marcha:

```
ARRAY TEXT(arrUUID;0)
ARRAY TEXT(arrStart;0)
ARRAY LONGINT(arrDuration;0)
ARRAY TEXT(arrInfo;0)

Repeat
   GET ACTIVITY SNAPSHOT(arrUUID;arrStart;arrDuration;arrInfo)
   If(Size of array(arrUUID)>0)
      TRACE  // llamada del depurador
   End if
Until(False)  // Bucle infinito
```

Obtiene arrays del tipo:

Expresión	Valor
arrUUID	4 elementos
arrUUID	0
arrUUID{0}	""
arrUUID{1}	"B289B998954F624CA1F175A309D32801"
arrUUID{2}	"35E516CF18E5A240A6CD8FB3E55FDEB4"
arrUUID{3}	"1EDC47A7FAE14646B7171F5322E15DAD"
arrUUID{4}	"15BA406DE0D31040B76E90FD33AB6FFA"
arrStart	4 elementos
arrStart	0
arrStart{0}	""
arrStart{1}	"10/12/2013 - 10:33:34"
arrStart{2}	"10/12/2013 - 10:33:38"
arrStart{3}	"10/12/2013 - 10:33:43"
arrStart{4}	"10/12/2013 - 10:33:37"
arrDuration	4 elementos
arrDuration	0
arrDuration{0}	0
arrDuration{1}	45779
arrDuration{2}	41806
arrDuration{3}	37303
arrDuration{4}	43208
arrInfo	4 elementos
arrInfo	0
arrInfo{0}	""
arrInfo{1}	"Escritura de datos (18 M/s)"
arrInfo{2}	"Selección a array: 10184 de 1340090"
arrInfo{3}	"Búsqueda de índices: 3004 de 8052 páginas"
arrInfo{4}	"Eliminación de registros: 3172 de 34128"

GET MACRO PARAMETER

GET MACRO PARAMETER (selector ; paramText)

Parámetro	Tipo		Descripción
selector	Entero largo	➡	Selección a utilizar
paramText	Texto	⬅	Texto devuelto

Descripción

*El comando **GET MACRO PARAMETER** devuelve, en el parámetro paramText, una parte o la totalidad del texto del método desde el cual se llama.*

El parámetro selector permite definir el tipo de información a recuperar. Puede pasar una de las siguientes constantes, del tema "":

Constante	Tipo	Valor
Full method text	Entero largo	1
Highlighted method text	Entero largo	2

*Si pasa **Full method text** en selector, todo el texto del método se devolverá en paramText. Si pasa **Highlighted method text** en selector, únicamente el texto seleccionado en método se devolverá en paramText.*

Ejemplo

*Consulte el ejemplo del comando **SET MACRO PARAMETER**.*

LAUNCH EXTERNAL PROCESS

LAUNCH EXTERNAL PROCESS (nomArchivo {; flujoEntrada {; flujoSalida {; flujoError}}}{; pid})

Parámetro	Tipo		Descripción
nomArchivo	Cadena	➡	Ruta de acceso y argumentos del archivo a abrir
flujoEntrada	Cadena, BLOB	➡	Flujo de entrada(stdin)
flujoSalida	Cadena, BLOB	⬅	Flujo de salida (stdout)
flujoError	Cadena, BLOB	⬅	Flujo de error(stderr)
pid	Entero largo	⬅	Identificador único del proceso externo

Descripción

*El comando **LAUNCH EXTERNAL PROCESS** permite iniciar un proceso externo de 4D, bajo Mac OS X y Windows.*
Bajo Mac OS X, este comando ofrece acceso a todas las aplicaciones ejecutables que puedan ser iniciadas desde el Terminal.

Pase en el parámetro nomArchivo la ruta de acceso de la aplicación a ejecutar, como también los argumentos requeridos (si es necesario).

*Bajo Mac OS X, puede igualmente pasar únicamente el nombre de la aplicación a ejecutar; 4D utilizará entonces la variable del entorno **PATH** para ubicar el ejecutable.*

***Advertencia:** este comando sólo puede iniciar aplicaciones ejecutables; no puede ejecutar instrucciones que hagan parte del shell (interprete de comandos). Por ejemplo, bajo Mac OS no es posible utilizar este comando para ejecutar la instrucción echo o las indirecciones.*

El parámetro flujoEntrada (opcional) contiene el stdin del proceso externo. Una vez el comando haya sido ejecutado, los parámetro flujoSalida y flujoError (si se pasan) devuelven respectivamente el stdout y el tderr del proceso externo. Puede utilizar los parámetros de tipo BLOB en lugar de las cadenas de caracteres si maneja datos binarios (como imágenes).

***Nota:** si utiliza la variable del entorno _4D_OPTION_BLOCKING_EXTERNAL_PROCESS vía el comando **SET ENVIRONMENT VARIABLE** (ejecución asincrónica), los parámetros flujoSalida y flujoError no se devuelven.*

Cuando se pasa, el parámetro pid (entero largo) devuelve el ID único del proceso creado para lanzar el comando, independientemente del estado de la opción _4D_OPTION_BLOCKING_EXTERNAL_PROCESS. Con esta información, es más fácil interactuar con los procesos externos creados por el comando, por ejemplo. para detenerlo. Si el lanzamiento del proceso falla, no se devuelve el parámetro pid.

Ejemplos bajo Mac OS X

Los siguientes ejemplos utilizan el Terminal Mac OS X, disponible en la carpeta Aplicaciones/Utilidades.

1. Para modificar los accesos a un archivo (chmod es el comando Mac OS X utilizado para modificar el acceso a los archivos):

```
LAUNCH EXTERNAL PROCESS("chmod +x /carpeta/miarchivo.txt")
```

2. Para editar un archivo de tipo texto (cat es el comando Mac OS X utilizado para editar los archivos). En este ejemplo, se pasa la ruta de acceso completa del comando:

```
C_TEXT(entrada;salida)
entrada:=""
LAUNCH EXTERNAL PROCESS("/bin/cat /carpeta/miarchivo.txt";entrada;salida)
```

3. Para obtener los contenidos de la carpeta "Usuarios" (ls -l es el equivalente Mac OS X del comando dir en DOS):

```
C_TEXT($In;$Out)
LAUNCH EXTERNAL PROCESS("/bin/ls -l /Usuarios";$In;$Out)
```

*4. Para iniciar una aplicación "gráfica" independiente, es preferible utilizar el comando sistema open (en este caso, la instrucción **LAUNCH EXTERNAL PROCESS** tiene el mismo efecto que hacer doble clic en la aplicación):*

```
LAUNCH EXTERNAL PROCESS("open /Applications/Calculator.app")
```

Ejemplos bajo Windows

5. Para abrir NotePad:

```
LAUNCH EXTERNAL PROCESS("C:\\WINDOWS\\notepad.exe")
```

6. Para abrir Notepad y abrir un documento específico:

```
LAUNCH EXTERNAL PROCESS("C:\\WINDOWS\\notepad.exe C:\\Docs\\nueva carpeta\\res.txt")
```

7. Para iniciar la aplicación Microsoft® Word® y abrir un documento específico (Note el uso de las dos ""):

```
$midoc:="C:\\Program Files\\Microsoft Office\\Office10\\WINWORD.EXE \"C:\\Documents and
Settings\\Macros\\Escritorio\\MisDocs\\Nuevacarpeta\\test.xml\""
LAUNCH EXTERNAL PROCESS($midoc;$tIn;$tOut)
```

8. Para ejecutar un script Perl (es necesario ActivePerl):

```
C_TEXT($entrada;$salida)
SET ENVIRONMENT VARIABLE("mivariable";"valor")
LAUNCH EXTERNAL PROCESS("D:\\Perl\\bin\\perl.exe D:\\Perl\\eg\\cgi\\env.pl";$entrada;$salida)
```

9. Para iniciar un comando con el directorio actual y sin mostrar la consola:

```
SET ENVIRONMENT VARIABLE("_4D_OPTION_CURRENT_DIRECTORY";"C:\\4D_VCS")
SET ENVIRONMENT VARIABLE("_4D_OPTION_HIDE_CONSOLE";"true")
LAUNCH EXTERNAL PROCESS("micomando")
```

10. Para permitir al usuario abrir un documento externo en Windows:

```
$nomdoc:=Select document("";"*.*";"Elija el archivo a abrir";0)
If(OK=1)
   SET ENVIRONMENT VARIABLE("_4D_OPTION_HIDE_CONSOLE";"true")
   LAUNCH EXTERNAL PROCESS("cmd.exe /C start \"\" \""+document+"\"")
End if
```

11. Los siguientes ejemplos recuperan la lista de procesos bajo Windows:

```
C_LONGINT($pid)
C_TEXT($stdin;$stdout;$stderr)

LAUNCH EXTERNAL PROCESS("tasklist";$pid) //gets PID only
LAUNCH EXTERNAL PROCESS("tasklist";$stdin;$stdout;$stderr;$pid) //obtener toda la información
```

Variables y conjuntos del sistema

Si el comando ha sido ejecutado correctamente, la variable sistema OK toma el valor 1. De lo contrario (archivo no encontrado, memoria insuficiente, etc.), toma el valor 0.

Load 4D View document

Load 4D View document (4DViewDocument) -> Resultado

Parámetro	Tipo		Descripción
4DViewDocument	BLOB	➡	Documento 4D View
Resultado	Objeto	⮌	Representación del objeto del documento 4D View

Descripción

*El comando **Load 4D View document** permite convertir un documento 4D View en un objeto 4D.*

Ni una licencia válida 4D View, ni una instancia del plug-in 4D View heredado en su entorno son necesarios para este comando.

Pase en el parámetro 4DViewDocument una variable BLOB o campo que contenga el documento 4D View a convertir. El comando devuelve un objeto 4D que describe toda la información almacenada originalmente en el documento 4D View, incluyendo:

- *estructura del documento (número de filas y columnas), tipo e información (versión, título...)*
- *atributos de celda (tipo de celda, valor, fórmula, nombre, estilo, seguridad...)*
- *atributos de columna (ancho, estilo, tipo, seguridad, visibilidad, ruptura...)*
- *atributos de fila (altura, estilo, tipo, seguridad, visibilidad, ruptura...)*
- *estilos, bordes y paneles*

Utilizando este comando, puede recuperar toda información almacenada en sus documentos 4D View y manejarlos en un formato abierto.

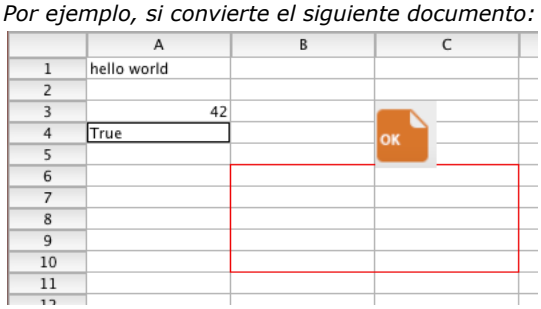
***Nota**: si necesita convertir documentos de 4D View a 4D View Pro, se recomienda utilizar el comando dedicado **VP Convert from 4D View** que realiza una conversión directa y transparente.*

Ejemplo

Desea cargar y convertir un documento 4D View almacenado en el disco:

```
C_BLOB($blob)
C_OBJECT($object)
DOCUMENT TO BLOB("document.4PV";$blob)
$object:=Load 4D View document($blob)
ALERT("Document title is "+$object.title)
```

Por ejemplo, si convierte el siguiente documento:



Obtendrá el siguiente resultado (objeto stringified):


```
{ "version": 9, "title": "4D View test", "subject": "", "author": "", "company": "", "note": "", "creationDate": "2017-06-13", "creationTime": 63230, "modificationDate": "2017-06-13", "modificationTime": 63295, "columnCount": 2048, "rowCount": 65535, "columnHeaderHeight": 380, "rowHeaderWidth": 1180, "columnWidth": 2160, "rowHeight": 320, "noExternalCall": false, "columns": [], "rows": [], "cells": [ { "kind": "value", "value": "hello world", "valueType": "string", "column": 1, "row": 1 }, { "kind": "value", "value": 42, "valueType": "real", "column": 1, "row": 3 }, { "kind": "value", "value": true, "valueType": "bool", "column": 1, "row": 4 } ], "cellNames": [], "customFormats": [], "rowEdges": [ { "style": 13, "color": 15597568, "left": 2, "top": 6, "right": 3, "bottom": 6 }, { "style": 13, "color": 15597568, "left": 2, "top": 11, "right": 3, "bottom": 11 } ], "columnEdges": [ { "style": 13, "color": 15597568, "left": 2, "top": 6, "right": 2, "bottom": 10 }, { "style": 13, "color": 15597568, "left": 4, "top": 6, "right": 4, "bottom": 10 } ], "defaultStyle": { "locked": false, "hidden": false, "gridHidden": false, "spellCheck": false, "pictHeights": false, "inputFilter": 0, "backColorEven": 16777215, "backColorOdd": 16777215, "fontID": 2, "fontSize": 11, "fontBold": false, "fontItalic": false, "fontUnderline": false, "fontOutline": false, "fontShadow": false, "fontCondensed": false, "fontExtended": false, "normalColorEven": 0, "normalColorOdd": 0, "zeroColorEven": 255, "zeroColorOdd": 255, "minusColorEven": 16711680, "minusColorOdd": 16711680, "hAlign": 0, "vAlign": 0, "rotation": 0, "wordWrap": false, "forceTextFormat": false, "numericFormat": 0, "stringFormat": 0, "booleanFormat": 0, "dateTimeFormat": 0, "pictureFormat": 0 }, "exportRanges": [], "fontNames": [ { "id": 2, "name": "Lucida Grande" } ], "inputFilters": [], "pictures": [ { "column": 3, "row": 3, "width": 920, "height": 1000, "drawingMode": 5, "behind": false, "fixedSize": false, "locked": false, "hOffset": 0, "vOffset": 0, "picture": "[object Picture]" } ] }
```


***Nota:** para más información sobre el formato del objeto, devuelto, contacte los servicios técnicos de 4D.*

OPEN URL

OPEN URL (ruta {; nomAp}{; *})

Parámetro	Tipo		Descripción
ruta	Cadena	➡	Ruta del documento o URL a abrir
nomAp	Cadena	➡	Nombre de la aplicación a utilizar
*	Operador	➡	Si se especifica = la URL no está traducida, Si se omite = la URL está traducida

Descripción

*El comando **OPEN URL** abre el archivo o URL pasado en el parámetro ruta con la aplicación indicada en nomAp (si hay).*

*El parámetro ruta puede contener bien un URL estándar o una ruta de acceso de archivo. El comando acepta dos puntos (':') bajo Mac OS, barras oblicuas ('\') bajo Windows o un URL Posix que comience por archivo://. Si el parámetro nomAp se omite, 4D primero intenta interpretar el parámetro ruta como un nombre de ruta de archivo. Si este es el caso, 4D solicitará al sistema abrir el archivo utilizando la aplicación más apropiada (por ejemplo, un navegador para los archivos .html, Word para los archivos .doc, etc.). El parámetro * se ignora en este caso.*

Si el parámetro ruta contiene un URL estándar (protocolos mailto:, news:, http:, etc.), 4D lanza el navegador web por defecto y accede al URL. Si no hay navegador en los volúmenes conectados al ordenador, el comando no tiene efecto.

Cuando se pasa el parámetro nomAp , el comando interroga al sistema. Si se instala una aplicación con este nombre, que se inicia y el comando le pide que abra la dirección URL o el documento especificado.
En Windows, el mecanismo para el reconocimiento del nombre de la aplicación es la misma que el utilizado por el comando "Ejecutar" del menú Inicio. Por ejemplo, podría pasar:

"iexplore" para iniciar Internet Explorer.
"chrome" para iniciar Chrome (si está instalado)
"winword" para iniciar MS Word (si está instalado)

***Nota**: encontrará la lista de aplicaciones instaladas en el registry en la siguiente llave:*
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths

Bajo OS X, el mecanismo utiliza el Finder que indexa automáticamente todas las aplicaciones instaladas. Puede reconocer toda aplicación .app por su nombre de package (con o sin el sufijo .app). Por ejemplo, puede pasar:

"safari"
"FireFox"
"TextEdit"

Si la aplicación nomAp no se encuentra, ningún error se devuelve; el comando se ejecuta como si el parámetro no hubiera sido especificado.

*4D codifica automáticamente los caracteres especiales de la URL. Si pasa el carácter *, 4D no traducirá los caracteres especiales del URL. Esta opción le permite acceder y enviar las URLS de tipo "http://www.server.net/page.htm?q=something".*

***Nota:** este comando no funciona cuando se llama desde un proceso web.*

Ejemplo 1

Los siguientes ejemplos muestran los diferentes tipos de cadenas que son aceptadas como URLs por el comando:

```
OPEN URL("http://www.4d.com")
OPEN URL("file://C:/Users/Laurent/Documents/pending.htm")
OPEN URL("C:\\Users\\Laurent\\Documents\\pending.htm")
OPEN URL("mailto:jean_martin@4d.fr")
```

Ejemplo 2

Este ejemplo puede utilizarse para lanzar la aplicación más conveniente:

```
$file:=Select document("";"";0)
If(OK=1)
   OPEN URL(Document)
End if
```

Ejemplo 3

Puede abrir un mismo archivo texto con diferentes aplicaciones utilizando el parámetro nomAp:

```
OPEN URL("C:\\temp\\cookies.txt") //abre el archivo con el Bloc de notas
OPEN URL("C:\\temp\\cookies.txt";"winword") //abre el archivo con MS Word (si está instalado)
OPEN URL("C:\\temp\\cookies.txt";"excel") //abre el archivo con MS Excel (si está instalado)
```

PROCESS 4D TAGS

PROCESS 4D TAGS (plantillaEntrada ; datosSalida {; param}{; param2 ; ... ; paramN})

Parámetro	Tipo		Descripción
plantillaEntrada	Texto, BLOB	➡	Datos que contiene las etiquetas a tratar
datosSalida	Texto, BLOB	⬅	Datos procesados
param	Expresión	➡	Parámetros pasados a la plantilla en ejecución

Descripción

*El comando **PROCESS 4D TAGS** provoca el procesamiento de las etiquetas de transformación 4D contenidas en el parámetro plantillaEntrada (campo o variable de tipo Texto o BLOB) mientras opcionalmente inserta valores utilizando los valores vía los parámetros param y devuelve los datos resultantes en datosSalida.Para una descripción completa de estas etiquetas, consulte la sección **Etiquetas de transformación 4D**.*

Este comando permite ejecutar un texto de tipo "template" que contiene las etiquetas y las referencias a las expresiones o variables 4D y produce un resultado dependiendo del contexto de ejecución y/o de los valores pasados en parámetro.

*Por ejemplo, puede utilizar este comando para generar y guardar las páginas HTML basadas en **páginas semi-dinámicas** que contienen las etiquetas de transformación 4D (sin que sea necesario que el servidor Web de 4D se inicie). Puede utilizarlas para enviar mensajes de correo electrónico en formato HTML que contengan procesamientos y/o referencias a los datos contenidos en la base a través de 4D Internet Commands. Es posible procesar cualquier tipo de datos basados en texto, tales como XML, SVG o texto multi-estilo.*

Pase los datos que contienen las etiquetas a procesar en el parámetro plantillaEntrada. Este parámetro puede ser un campo o una variable de tipo Texto o BLOB. El tipo Texto por lo general es suficiente (los parámetros pueden recibir hasta 2 GB de texto).

***Nota de compatibilidad**: a partir de la versión 12 de 4D, cuando utiliza parámetros de tipo BLOB, el comando considera automáticamente que el conjunto de caracteres utilizado por los BLOBs es MacRoman. Para mayor eficiencia, se recomienda utilizar los parámetros de tipo Texto para los cuales los procesos se efectúan en modo Unicode.*

Todas las etiquetas de transformación de 4D son soportadas (4DTEXT, 4DHTML, 4DSCRIPT, 4DLOOP, 4DEVAL, etc.),

***Nota:** en caso de utilizar la etiqueta 4DINCLUDE fuera del marco del servidor web (proceso web):*

- *Con 4D en modo local o 4D Server, la carpeta por defecto es la carpeta que contiene el archivo de estructura de la base,*
- *Con 4D en modo remoto, la carpeta por defecto es la carpeta que contiene la aplicación 4D.*

*El comando **PROCESS 4D TAGS** soporta un número indefinido de parámetros param que se pueden insertar en el código que se ejecuta. Al igual que con los métodos proyecto, dichos parámetros pueden contener valores escalares de tipos variados (texto, fecha, hora, entero largo, real, etc), así como también objetos o colecciones. También puede utilizar los arrays, por intermedio de punteros de arrays. Dentro del código procesado por las etiquetas 4D, estos parámetros son accesibles por medio de argumentos estándar ($1, $2, etc.), al igual que en los métodos 4D (ver ejemplo).*
*Un conjunto dedicado de variables locales se define en el contexto de ejecución del comando **PROCESS 4D TAGS**. Estas variables pueden ser escritas o leídas durante el procesamiento.*

***Nota de compatibilidad**: en versiones anteriores de 4D, las variables locales definidas en el contexto de llamada se podían acceder en el contexto de ejecución de **PROCESS 4D TAGS** en modo interpretado. A partir de 4D v14 R4, este ya no es el caso.*

Después de la ejecución del comando, el parámetro datosSalida recibe los datos del parámetro plantillaEntrada, junto con el resultado del proceso de las etiquetas 4D que contiene, cuando aplica. Si datosEntrada no contiene las etiquetas 4D, el contenido de datosSalida es idéntico al de plantillaEntrada.

El parámetro datosSalida puede ser un campo o una variable, pero debe ser del mismo tipo que el parámetro plantillaEntrada.

***Nota**: este comando ya no llama al **Método base On Web Authentication**.*

Ejemplo 1

Este ejemplo carga un documento de tipo 'template', procesa las etiquetas que contiene y luego lo guarda:

```
C_BLOB($Blob_x)
C_BLOB($blob_out)
C_TEXT($inputText_t)
C_TEXT($outputText_t)

DOCUMENT TO BLOB("mytemplate.txt";$Blob_x)
$inputText_t:=BLOB to text($Blob_x;UTF8 text without length)
PROCESS 4D TAGS($inputText_t;$outputText_t)
TEXT TO BLOB($outputText_t;$blob_out;UTF8 text without length)
BLOB TO DOCUMENT($document;$blob_out)
```

Ejemplo 2

Este ejemplo genera un texto utilizando los datos de los arrays:

```
ARRAY TEXT($array;2)
$array{1}:="hello"
$array{2}:="world"
$input:="<!--#4DEVAL $1-->"
$input:=$input+"<!--#4DLOOP $2-->"
$input:=$input+"<!--#4DEVAL $2->{$2->}--> "
$input:=$input+"<!--#4DENDLOOP-->"
PROCESS 4D TAGS($input;$output;"elements = ";->$array)
 // $output = "elements = hello world"
```

SET ENVIRONMENT VARIABLE

SET ENVIRONMENT VARIABLE (nomVar ; valorVar)

Parámetro	Tipo		Descripción
nomVar	Cadena	➡	Nombre de la variable a definir
valorVar	Cadena	➡	Valor de la variable o "" para restablecer el valor por defecto

Descripción

*El comando **SET ENVIRONMENT VARIABLE** permite fijar el valor de una variable de entorno bajo Mac OS X y Windows. Está diseñado para utilizarse con el comando **LAUNCH EXTERNAL PROCESS**. También funciona con el comando **PHP Execute**.*

Pase el nombre de la variable a definir en nomVar y su valor en valorVar.

- *Para obtener la lista general de las variables de entorno y sus posibles valores, por favor consulte la documentación técnica de su sistema operativo.*
- *Para ver la lista de variables de entorno disponibles con el comando **LAUNCH EXTERNAL PROCESS**, consulte la documentación de este comando. Note que tres variables entorno específicas están disponibles para uso en este contexto:*

_4D_OPTION_CURRENT_DIRECTORY: permite definir el directorio actual del proceso externo a iniciar. En valorVar, debe pasar la ruta de acceso del directorio (sintaxis de tipo HFS en Mac OS y DOS en Windows).
_4D_OPTION_HIDE_CONSOLE (Windows únicamente): permite ocultar la ventana de la consola DOS. Debe pasar "true" en valorVar para ocultar la consola o "false" para mostrarla.
_4D_OPTION_BLOCKING_EXTERNAL_PROCESS: permite ejecutar el proceso externo en modo asincrónico, en otras palabras, sin bloqueo para las otras aplicaciones. Debe pasar "false" en valorVar para definir una ejecución asincrónica o "true" para una ejecución sincrónica (no es posible utilizar un valor por defecto con esta variable).

*Estas variables son válidas en el proceso actual para la siguiente llamada a **LAUNCH EXTERNAL PROCESS**.*

Ejemplo

*Consulte los ejemplos del comando **LAUNCH EXTERNAL PROCESS**.*

SET MACRO PARAMETER

SET MACRO PARAMETER (selector ; paramText)

Parámetro	Tipo		Descripción
selector	Entero largo	➡	Selección a utilizar
paramText	Texto	➡	Texto enviado

Descripción

*El comando **SET MACRO PARAMETER** inserta el texto paramText en el método desde el cual se llama.*
Si se ha seleccionado texto en el método, el parámetro selector permite definir si el texto paramText debe reemplazar todo el método o únicamente el texto seleccionado. En selector, puede pasar una de las siguientes constantes, del tema "":

Constante	Tipo	Valor
Full method text	Entero largo	1
Highlighted method text	Entero largo	2

Si ningún texto ha sido seleccionado, paramText se inserta en el método.

Nota

*Para que los comandos **GET MACRO PARAMETER** y **SET MACRO PARAMETER** funcionen correctamente, el nuevo atributo "version" debe declararse en la macro misma de esta forma:*

```
<macro name="MyMacro" version="2">
--- Text of macro ---
</macro>
```

Ejemplo

Esta macro crea un nuevo texto que será devuelto al método llamante:

```
C_TEXT($texto_entrada)
C_TEXT($texto_salida)
GET MACRO PARAMETER(Highlighted method text;$texto_entrada)
  `Suponga que el texto seleccionado es una tabla, ej. "[Clientes]"
$texto_salida:=""
$texto_salida:=$texto_salida+Command name(47)+"("+$texto_entrada+")" ` Selecionar todos ([Clientes])
$texto_salida:=$texto_salida+"$i:="+Command name(76)+"("+$texto_entrada+")" ` $i:=Records in selection([Clientes])
SET MACRO PARAMETER(Highlighted method text;$texto_salida)
  `Reemplaza el texto seleccionado por el nuevo código
```

Verify password hash

Verify password hash (contrasena ; hash) -> Resultado

Parámetro	Tipo		Descripción
contrasena	Cadena	➡	La contraseña de usuario. Sólo se utilizan los primeros 72 caracteres.
hash	Cadena	➡	Un hash de contraseña.
Resultado	Booleano	↩	Devuelve TRUE si la contraseña y hash coinciden, de lo contrario devuelve FALSE.

Descripción

*La función **Verify password hash** verifica que el hash dado coincida con la contrasena dada.*
*Esta función compara la contrasena con un hash generado por la función **Generate password hash**.*

Gestión de errores

*Se pueden devolver los errores siguientes. Puede revisar un error con los comandos **GET LAST ERROR STACK** y **ON ERR CALL**.*

Número	Mensaje
850	Password-hash: Algoritmo no soportado.
851	Password-hash: Fallo de verificación de consistencia.

Ejemplo

*Este ejemplo verifica un hash de contraseña creado previamente por **Generate password hash** y almacenado en una tabla [Users] con una contraseña introducida recientemente:*

```
C_TEXT($password)
$password:=Request("Por favor introduzca su contraseña")

If(Verify password hash($password;[Users]hash))
   ALERT("Contraseña correcta")
Else
   ALERT("Contraseña incorrecta")
End if
```

***Nota:** la contraseña nunca se almacena en el disco, sólo el hash. Utilizando una aplicación 4D remota, el hash podría ser producido del lado del cliente. Si en cambio, utiliza un front end basado en JavaScript (o similar), la mejor práctica para la seguridad es crear el hash del lado del servidor. Por supuesto, debe utilizar una conexión de red cifrada TLS para la seguridad, ya que esto requiere la transferencia de la contraseña a través de la red.*

Imágenes

- *Introducción a las imágenes*
- *BLOB TO PICTURE*
- *COMBINE PICTURES*
- *CONVERT PICTURE*
- *CREATE THUMBNAIL*
- *Equal pictures*
- *Get picture file name*
- *GET PICTURE FORMATS*
- *GET PICTURE FROM LIBRARY*
- *GET PICTURE KEYWORDS*
- *GET PICTURE METADATA*
- *Is picture file*
- *PICTURE CODEC LIST*
- *PICTURE LIBRARY LIST*
- *PICTURE PROPERTIES*
- *Picture size*
- *PICTURE TO BLOB*
- *READ PICTURE FILE*
- *REMOVE PICTURE FROM LIBRARY*
- *SET PICTURE FILE NAME*
- *SET PICTURE METADATA*
- *SET PICTURE TO LIBRARY*
- *TRANSFORM PICTURE*
- *WRITE PICTURE FILE*
- *_o_PICTURE TO GIF*
- *_o_PICTURE TYPE LIST*
- *_o_QT COMPRESS PICTURE*
- *_o_QT COMPRESS PICTURE FILE*
- *_o_QT LOAD COMPRESS PICTURE FROM FILE*
- *_o_SAVE PICTURE TO FILE*

Introducción a las imágenes

Formatos nativos soportados

4D integra una gestión nativa de los formatos de imagen. Esto significa que las imágenes se mostrarán y almacenarán en su formato original, sin interpretación en 4D. Las características específicas de los diferentes formatos (sombras, áreas transparentes, etc.) se conservan al copiar y pegar y se mostrarán sin alteración. Este soporte nativo es válido para todas las imágenes almacenadas en 4D: librería de imágenes, imágenes pegadas en formularios en entorno Diseño, imágenes pegadas en campos o variables en modo Aplicación, etc.

4D utiliza APIs nativos para codificar y decodificar imágenes (campos y variables) bajo Windows y Mac OS.
Estas implementaciones ofrecen acceso a varios formatos nativos, incluyendo el formato RAW, actualmente utilizado en cámaras digitales.

- ***Bajo Windows**, 4D utiliza WIC (Windows Imaging Component). WIC soporta nativamente los siguientes formatos: BMP, PNG, ICO (decodificación únicamente), JPEG, GIF, TIFF y WDP (Microsoft Windows Digital Photo).*
  Es posible utilizar formatos adicionales tales como JPEG-2000 instalando codecs WIC de terceros.
- ***Bajo Mac OS**, 4D utiliza ImageIO. Todos los codecs ImageIO disponibles son por lo tanto soportados nativamente para codificación (lectura) como también para decodificación (escritura):*

Decodificación	***Codificación***
public.jpeg	*public.jpeg*
com.compuserve.gif	*com.compuserve.gif*
public.png	*public.png*
public.jpeg-2000	*public.jpeg-2000*
com.nikon.raw-image	*public.tiff*
com.pentax.raw-image	*com.adobe.photoshop.image*
com.sony.arw-raw-image	*com.adobe.pdf*
com.adobe.raw-image	*com.microsoft.bmp*
public.tiff com.canon.crw-raw-image	*com.truevision.tga-image*
com.canon.cr2-raw-image	*com.sgi.sgi-image*
com.canon.tif-raw-image	*com.apple.pict (obsoleto)*
com.sony.raw.image	*com.ilm.openexr-image*
com.olympus.raw-image	
com.konicaminolta.raw-image	
com.panasonic.raw-image	
com.fuji.raw-image	
com.adobe.photoshop-image	
com.adobe.illustrator.ai-image	
com.adobe.pdf	
com.microsoft.ico	
com.microsoft.bmp	
com.truevision.tga-image	
com.sgi.sgi-image	
com.apple.quicktime-image (obsoleto)	
com.apple.icns	
com.apple.pict (obsoleto)	
com.apple.macpaint-image	
com.kodak.flashpix-image	
public.xbitmap-image	
com.ilm.openexr-image	
public.radiance	

*Tanto en Windows como en Mac OS, los formatos soportados varían en función del sistema operativo y de los codecs personalizados instalados en las máquinas. Para conocer los codecs disponibles, debe utilizar el comando **PICTURE CODEC LIST**.*

***Nota:** WIC e ImageIO permiten el uso de metadatos en las imágenes. Dos comandos, **SET PICTURE METADATA** y **GET PICTURE METADATA**, le permiten beneficiarse de metadatos en sus desarrollos.*

Identificaciones de los códigos de imágenes

*Los formatos de imágenes reconocidos por 4D son devueltos por el comando **PICTURE CODEC LIST** como identificadores de codecs de imágenes. Estos identificadores pueden ser:*

- *una extensión (por ejemplo ".gif")*
- *un tipo Mime (por ejemplo "image/jpeg")*

La forma utilizada para cada formato depende del modo de declaración del codec a nivel del sistema operativo.
*La mayoría de los comandos de gestión de imágenes de 4D pueden recibir un identificador de codec como parámetro. Por lo tanto es imperativo utilizar el identificador sistema devuelto por el comando **PICTURE CODEC LIST**.*

Formato de imagen no disponible

Un icono específico se muestra para las imágenes guardadas en un formato que no está disponible en la máquina. La extensión del formato de falta se muestra en la parte inferior del icono:

El icono se utiliza de forma automática siempre que la imagen sea para mostrar:

FirstName :	LastName :	Photo :
Elizabeth	Smith	
Gerry	Mc Namara	
Henry	Portier	

Este icono indica que la imagen no se puede mostrar o manipular de forma local, pero puede guardar sin alteración para que se pueda mostrar en otras máquinas. Este es el caso, por ejemplo, para las imágenes PDF en Windows, o para las imágenes basadas en PICT mostradas un 4D Server 64 bits bajo OS X.

Activación de QuickTime (compatibilidad)

Por defecto, los codecs de imagen relacionados con QuickTime ya no son soportados en 4D a partir de la v14.

*Por razones de compatibilidad, puede reactivar QuickTime en su aplicación por medio de la opción QuickTime support del comando **SET DATABASE PARAMETER**. Sin embargo, ya no recomendamos utilizar codecs QuickTime.*

***Nota:** la opción de reactivación de QuickTime se ignora en las versiones 64 bits de 4D Developer Edition (sin soporte QuickTime).*

Coordenadas de un clic sobre una imagen

4D permite recuperar las coordenadas locales de un clic en un campo o una variable imagen, incluso si se ha aplicado un desplazamiento o un zoom a la imagen. Este mecanismo, similar al de una imagen map, puede ser utilizado, por ejemplo, para manejar barras de botones desplazables o la interfaz de software de cartografía.

*Las coordenadas del clic se devuelven en MouseX y MouseY **Variables sistema**. Las coordenadas son expresadas en píxeles con respecto a la esquina superior izquierda de la imagen (0,0). Si el ratón está fuera del sistema de coordenadas de la imagen, el valor -1 se devuelve en MouseX y MouseY.*

Debe obtener el valor de estas variables como parte de un evento de formulario On Clicked, On Double Clicked, On Mouse up, On Mouse Enter, o On Mouse Move.

Operadores sobre imágenes

*4D le permite efectuar operaciones sobre imágenes 4D, tales como concatenación, superposición, etc. Este punto se trata en la sección **Operadores de imágenes**.*

BLOB TO PICTURE

BLOB TO PICTURE (blobImag ; imagen {; codec})

Parámetro	Tipo		Descripción
blobImag	BLOB	➡	BLOB contiene una imagen
imagen	Imagen	⬅	Campo o variable imagen 4D
codec	Cadena	➡	Identificador de codec de imagen

Descripción

El comando **BLOB TO PICTURE** inserta una imagen almacenada en un BLOB en un campo o variable imagen 4D, sin importar su formato original.

Este comando es similar al comando **READ PICTURE FILE**, simplemente se aplica a un BLOB en lugar de a un archivo. Esto permite mostrar imágenes almacenadas en formato nativo en los BLOBs. Puede cargar una imagen en un BLOB utilizando, por ejemplo, el comando **DOCUMENT TO BLOB** o **PICTURE TO BLOB**.

En el parámetro blobImag se pasa el campo o imagen BLOB que contiene la imagen. La imagen puede estar en cualquier formato soportado nativamente por 4D. Puede obtener la lista de formatos disponibles utilizando el comando **PICTURE CODEC LIST**. Si pasa el parámetro opcional codec, 4D utilizará el valor en este parámetro para decodificar el BLOB (ver el funcionamiento específico del comando con este tercer parámetro a continuación).

Pase en el parámetro imagen la variable o el campo 4D de tipo imagen el cual debe mostrar la imagen.

Nota: el formato interno de la imagen se conserva dentro de la variable o campo 4D.

Después de la ejecución del comando, imagen contiene la imagen a mostrar en 4D.

El parámetro opcional codec le permite especificar el codec a utilizar para la decodificación del BLOB.

Si pasa en codec un codec reconocido por 4D (devuelto por el comando **PICTURE CODEC LIST**), se aplica al BLOB y la imagen se devuelve en el campo o variable imagen.

Si pasa en codec un codec no reconocido por 4D, un nuevo codec se registra dinámicamente con el identificador pasado en el parámetro. 4D devuelve una imagen que encapsula el BLOB y la variable OK toma el valor 1. En este caso, para recuperar el BLOB deberá utilizar el comando **PICTURE TO BLOB** con el mismo identificador personalizado. Este mecanismo en particular puede utilizarse para cumplir con dos necesidades específicas:

- encapsulación de un BLOB (que no es una imagen) en una imagen,
- carga de una imagen sin utilizar un codec.

La implementación de estos mecanismos permite, más específicamente, la creación de "arrays de BLOBs " vía arrays de imagen. Este técnica debe utilizarse con precaución porque como los arrays se cargan completamente en la memoria, trabajar con BLOBs de gran tamaño puede afectar el funcionamiento de la aplicación.

Nota: un BLOB creado por el comando **VARIABLE TO BLOB** se administra automáticamente; no es necesario pasar un codec para encapsularlo ya que el BLOB está "firmado". En este caso, para la operación contraria, deberá pasar ".4DVarBlog" como identificador de codec al comando **PICTURE TO BLOB**.

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. Si la conversión ha fallado (QuickTime no está instalado, el BLOB no contiene una imagen legible, parámetro codec reconocido pero no validado por el BLOB , etc.), OK toma el valor 0 y el campo o variable imagen 4D se devuelve vacío.

COMBINE PICTURES

COMBINE PICTURES (imagenResult ; imag1 ; operador ; imag2 {; despHor ; despVert})

Parámetro	Tipo		Descripción
imagenResult	Imagen	⇐	Imagen resultante de la combinación
imag1	Imagen	⇒	Primera imagen a combinar
operador	Entero largo	⇒	Tipo de combinación a realizar
imag2	Imagen	⇒	Segunda imagen a combinar
despHor	Entero largo	⇒	Desplazamiento horizontal para la superposición
despVert	Entero largo	⇒	Desplazamiento vertical para la superposición

Descripción

*El comando **COMBINE PICTURES** permite combinar las imágenes imag1 e imag2 en modo operador para producir una tercera, imagenResult. La imagen resultante es de tipo compuesto y conserva todas las características de las imágenes fuente.*

***Nota:** este comando extiende las funcionalidades ofrecidas por los operadores clásicos de transformación de imágenes (+/, etc., ver la sección **Operadores de imágenes**). Estos operadores permanecen totalmente utilizables en 4D v11.*

*En operador, pase el tipo de combinación a aplicar. Se proponen tres tipos de combinaciones, accesibles a través de las constantes del tema "**Picture Transformation**":*

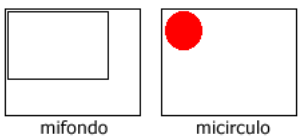
- ***Horizontal concatenation** (1): imag2 está asociada a imag1, la esquina superior izquierda de imag2 coincide con la esquina superior derecha de imag1.*
- ***Vertical concatenation** (2): imag2 está asociada a imag1, la esquina superior izquierda de imag2 coincide con la izquina inferior izquierda de imag1.*
- ***Superimposition** (3): imag2 está ubicada sobre pict1, la esquina superior izquierda de imag2 coincide con la esquina superior izquierda de imag1.*

Si se utilizan los parámetros opcionales despHor y despVert, una traducción se aplica a imag2 antes de la superposición. Los valores pasados en despHor y despVert deben estar en píxeles. Pase valores positivos para un desplazamiento a la derecha o hacia abajo y un valor negativo para un desplazamiento a la izquierda o hacia arriba.

***Nota:** la superposición efectuada por el comando **COMBINE PICTURES** difiere de la superposición propuesta por los operadores clásicos & y |(superposición exclusiva y superposición inclusiva). Mientras que el comando **COMBINE PICTURES** conserva las características de cada imagen fuente en la imagen resultantes, los operadores & y | procesan cada píxel y generan una imagen bitmap en todos los casos. Estos operadores, concebidos originalmente para las imágenes monocromáticas, ahora son obsoletos.*

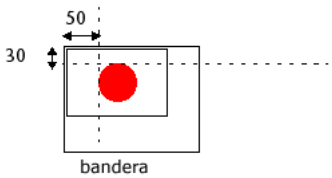
Ejemplo

Dadas las siguientes imágenes:




```
COMBINE PICTURES(bandera;mifondo;Superimposition;micirculo;50;30)
```

Resultado:



CONVERT PICTURE

CONVERT PICTURE (imagen ; codec {; compresion})

Parámetro	Tipo		Descripción
imagen	Imagen	➡	Imagen a convertir
		⬅	Imagen convertida
codec	Cadena	➡	Identificador de codec de imagen
compresion	Real	➡	Calidad de compresión

Descripción

*El comando **CONVERT PICTURE** convierte imagen en un nuevo tipo.*

El parámetro codec indica el tipo de imagen a generar. Un codec puede ser una extensión (por ejemplo, ".gif") o un tipo Mime (por ejemplo, "image/jpg").
*Puede obtener una lista de codecs disponibles utilizando el comando **PICTURE CODEC LIST**.*

Si el campo o variable imagen es de tipo compuesto (si por ejemplo es el resultado de la acción copiar -pegar), sólo la información correspondiente al tipo codec se conserva en la imagen resultante.

***Nota:** si el tipo de codec solicitado es el mismo que el tipo original de la imagen, no se efectúa ninguna conversión y la imagen se devuelve "tal cual" (excepto cuando se utiliza el parámetro compresion, ver a continuación).*

El parámetro opcional compresion, si se pasa, permite definir la calidad de compresión a aplicar a la imagen resultante cuando se utiliza un Codec compatible. En compresion, pase un valor entre 0 y 1 para especificar la calidad de la compresión, donde 0 es la calidad más mediocre (alta compresión) y 1 la mejor calidad (compresión baja). Este parámetro sólo se tiene en cuenta cuando el codec soporta la compresión (por ejemplo JPEG o HDPhoto) y es soportado por los APIs WIC y ImageIO.
*Para mayor información sobre los APIs de gestión de imagen en 4D, consulte la sección **Introducción a las imágenes**. Por defecto, si omite el parámetro compresion, se aplica la mejor calidad (compresion =1).*

***Nota:** si desea llamar a **CONVERT PICTURE** con un tipo de imagen que no es compatible con las versiones 64 bits de 4D (como PICT), asegúrese de que la conversión se realice en una versión 32 bits de 4D, donde el tipo original es soportado. Para más información, consulte la página **Pasar de 32 bits a 64 bits**.*

Ejemplo 1

Conversión de la imagen vpFoto al formato jpeg:

```
CONVERT PICTURE(vpFoto;".jpg")
```

Ejemplo 2

Conversión de una imagen con calidad del 60%:

```
CONVERT PICTURE(vPicture;".JPG";0.6)
```

CREATE THUMBNAIL

CREATE THUMBNAIL (fuente ; dest {; ancho {; altura {; modo {; profundidad}}}})

Parámetro	Tipo		Descripción
fuente	Imagen	➡	Campo o variable imagen 4D a convertir en miniatura
dest	Imagen	⬅	Miniatura resultante
ancho	Entero	➡	Largo de la miniatura en píxeles, Valor por defecto = 48
altura	Entero	➡	Alto de la miniatura en píxeles, Valor por defecto = 48
modo	Entero	➡	Modo de creación de la miniatura Valor por defecto = Proporcional centrado (6)
profundidad	Entero	➡	Colores de la miniatura en bits/píxeles Valor por defecto = Profundidad de pantalla actual (0)

Descripción

*El comando **CREATE THUMBNAIL**devuelve una miniatura a partir de una imagen fuente. Las miniaturas se utilizan generalmente para la previsualización de imágenes en software multimedia o sitios web.*

Pase en el parámetro fuente la variable o el campo imagen 4D que contiene la imagen a reducir en forma de miniatura y en el parámetro dest el campo o variable imagen 4D que debe recibir la miniatura resultante.

Los parámetros opcionales largo y alto definen el tamaño en píxeles de la miniatura. si omite estos parámetros, el tamaño por defecto de la miniatura será de 48 x 48 píxeles.

*El parámetro opcional modo define el modo de creación de la miniatura, es decir la manera en que será redimensionada. Hay tres modos disponibles. Las siguientes constantes predefinidas son suministradas por 4D en el tema **Formatos de salida de imágenes**:*

Constante	Tipo	Valor
Scaled to fit	Entero largo	2
Scaled to fit proportional	Entero largo	5
Scaled to fit prop centered	Entero largo	6

***Nota:** sólo estas tres constantes pueden ser utilizadas con **CREATE THUMBNAIL**. Las otras constantes en este tema no pueden aplicarse a este comando.*

Si no introduce ningún parámetro, el modo 6 "Scaled to fit prop centered" se aplica por defecto. A continuación se ilustran los diferentes modos:

Imagen fuente

Miniaturas resultantes (48x48)

- *Scaled to fit = 2*

- *Scaled to fit proportional = 5*

- *Scaled to fit prop centered = 6 (modo por defecto)*

***Nota:** con los modos "Scaled to fit proportional" y "Scaled to fit prop centered", los espacios vacíos aparecerán en blanco. Cuando estos modos se aplican a campos o imágenes en formularios 4D, el espacio libre es transparente.*

El parámetro profundidad se ignora y por lo tanto debe omitirse. El comando siempre utiliza la profundidad de pantalla actual (número de colores).

El parámetro opcional profundidad define el número de colores bajo Mac OS (es decir, la profundidad de pantalla) a conservar en la miniatura resultante. El parámetro es un entero igual al número de bits por píxel: 1, 2, 4, 8, 16 ó 32. Introduzca 0 para utilizar la profundidad de pantalla actual (valor por defecto).

***Nota:** bajo Windows, se ignora el parámetro profundidad; el comando siempre utiliza la profundidad de pantalla actual.*

Equal pictures

Equal pictures (imagen1 ; imagen2 ; mascara) -> resultado

Parámetro	Tipo		Descripción
imagen1	Campo imagen, Variable imagen	➡	Imagen fuente original
imagen2	Campo imagen, Variable imagen	➡	Imagen a comparar
mascara	Campo imagen, Variable imagen	⬅	Máscara resultante
resultado	Booleano	↩	True si ambas imágenes son idénticas; de lo contrario, False

Descripción

*El comando **Equal pictures** compara con precisión las dimensiones y el contenido de dos imágenes.*

Pase en imagen1 la imagen fuente y en imagen2 una imagen comparar con la imagen fuente.

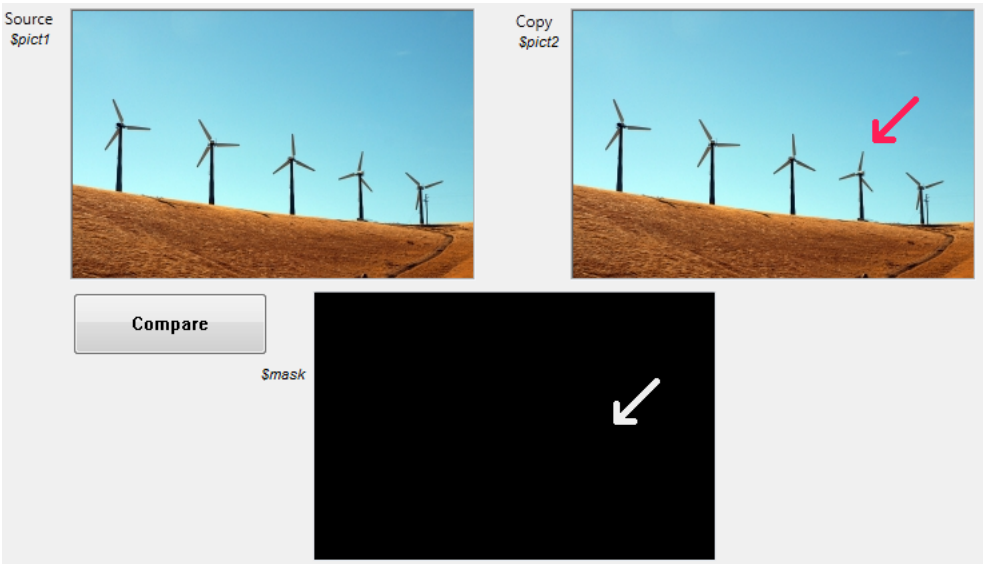
- *Si las imágenes no son de la misma dimensión, el comando devuelve **False** y el parámetro mascara contiene una imagen vacía.*
- *Si las dos imágenes son del mismo tamaño pero tienen contenidos diferentes, el comando devuelve **False** y el parámetro mascara contiene la imagen mascara resultante de la comparación de las dos imágenes. Esta comparación se realiza píxel por píxel. Cada píxel diferente aparece en blanco sobre un fondo negro.*
- *Si las dos imágenes son idénticas, el comando devuelve **True** y el parámetro mascara contiene una imagen completamente negra.*

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si se comparan las dos imágenes. En caso de anomalía, particularmente si una de las imágenes no se inicializa (imagen vacía), la variable OK toma el valor 0.

Ejemplo

En el siguiente ejemplo, comparamos dos imágenes (pict1 y pict2) y mostramos la máscara resultante:



*Este es el código del botón **Compare**:*

```
$equal :=Equal pictures($pict1;$pict2;$mask)
```

Get picture file name

Get picture file name (imagen) -> Resultado

Parámetro	*Tipo*		*Descripción*
imagen	*Campo imagen, Variable imagen*	→	*Imagen para la cual obtener el nombre por defecto*
Resultado	*Texto*	↩	*Nombre por defecto del archivo imagen*

Descripción

El comando ***Get picture file name*** *devuelve el nombre por defecto actual de la imagen pasada como parámetro.*

El nombre por defecto se utiliza durante la exportación de la imagen en un archivo disco. Puede definirse automáticamente basado en el nombre original del archivo imagen importado en el campo o la variable imagen, o utilizando el comando ***SET PICTURE FILE NAME****. Para mayor información, consulte el manual de Diseño.*

Si la imagen no tienen un nombre por defecto, el comando devuelve una cadena vacía.

GET PICTURE FORMATS

GET PICTURE FORMATS (imagen ; codecIDs)

Parámetro	Tipo		Descripción
imagen	Imagen	➡	Campo o variable imagen a analizar
codecIDs	Array texto	⬅	IDs de codec Imagen

Descripción

Tema: Imágenes

El comando **GET PICTURE FORMATS** devuelve un array de todos los IDs codec (formatos imagen) contenidos en la imagen pasada como parámetro.Una imagen 4D (campo o variable) puede contener la misma imagen codificada en diferentes formatos, tales como PNG, BMP, GIF...

En el parámetro imagen, se pasa una variable imagen cuyos formatos desea que se devuelvan en el array codecIDs.

Los identificadores codec devueltos son definidos por 4D exactamente de la misma manera que para el comando **PICTURE CODEC LIST**PICTURE CODEC LIST. Pueden ser devueltos de las siguientes formas:

- Como extensiones (por ejemplo, ".gif")
- Como tipos Mime (por ejemplo, "image/jpeg")
- Como códigos QuickTime de 4 caracteres

Notas:

- Los siguientes codecs, manejados internamente por 4D, siempre se devuelven como extensiones: JPEG, PNG, TIFF, GIF, BMP, SVG, PDF, EMF.
- Los códigos QuickTime de 4 caracteres pueden ser devueltos en las bases de datos donde la opción de compatibilidad QuickTime support ha sido definida (utilizando el comando **SET DATABASE PARAMETER**SET DATABASE PARAMETER). Sin embargo, QuickTime ya no se soporta en 4D y no se recomienda el uso de codecs QuickTime.

Para más información sobre los IDs codec imagen, consulte la sección **Imágenes**.

Ejemplo

Usted quiere saber los formatos de imagen almacenados en un campo para el registro actual:

```
ARRAY TEXT($aTPictureFormats;0)
 //Obtener todos los formatos guardados
GET PICTURE FORMATS([Employees]Photo;$aTPictureFormats)
```

GET PICTURE FROM LIBRARY

GET PICTURE FROM LIBRARY (refImag | nomImag ; imagen)

Parámetro	Tipo		Descripción
refImag | nomImag	Entero largo, Cadena	➡	Número de referencia o nombre de una imagen de la librería de imágenes
imagen	Variable imagen	⬅	Imagen de la librería de imágenes

Descripción

El comando **GET PICTURE FROM LIBRARY** devuelve en el parámetro imagen la imagen de la librería de imágenes cuyo número de referencia se pasa en refImag o cuyo nombre se pasa en nomImag.
Si no hay una imagen con ese nombre o número de referencia, **GET PICTURE FROM LIBRARY** no modifica imagen.

Ejemplo 1

El siguiente ejemplo devuelve en vgMiImagen la imagen cuyo número de referencia se almacena en la variable local $vlRefImag:

```
GET PICTURE FROM LIBRARY($vlRefImag;vgMiImagen)
```

Ejemplo 2

El siguiente ejemplo devuelve en $DDcom_Prot_MiImagen la imagen con el nombre "DDcom_Prot_Boton1" almacenada en la librería de imágenes:

```
GET PICTURE FROM LIBRARY("XP - Aceptar";$XP_Aceptar)
```

Ejemplo 3

Ver el tercer ejemplo para el comando **PICTURE LIBRARY LIST**.

Variables y conjuntos del sistema

La variable sistema **OK** toma el valor 1 si la imagen existe en la librería de imágenes. De lo contrario, **OK** toma el valor cero.

Gestión de errores

Si no hay suficiente memoria para devolver la imagen, se genera el error -108. Puede interceptar este error utilizando un método de gestión de errores.

GET PICTURE KEYWORDS

GET PICTURE KEYWORDS (imagen ; arrayPalabrasClaves {; *})

Parámetro	Tipo		Descripción
imagen	Campo imagen, Variable imagen	➡	Imagen para la cual obtener las palabras claves asociadas
arrayPalabrasClaves	Array texto	⬅	Array con las palabras claves extraídas
*	Operador	➡	Si se pasa = usar valores distintos

Descripción

*El comando **GET PICTURE KEYWORDS** devuelve en el array arrayPalabrasClaves, la lista de palabras claves asociadas a la imagen pasada como parámetro.*

*Sólo se tienen en cuenta las palabras claves definidas vía los metadatos **IPTC/Keywords**. Otros tipos de metadatos son ignorados por el comando. El comando funciona únicamente con los tipos de imágenes que soportan este tipo de metadatos (JPEG, TIFF, etc.).*

***Nota**: los metadatos de tipo IPTC/Keywords ahora son indexables en 4D (ver el manual de Diseño).*

*Si pasa el parámetro *, el método sólo devuelve los "valores distintos" de palabras claves, es decir una lista sin duplicados.*

Si la imagen no contiene palabras claves o metadatos IPTC/Keywords, el comando devuelve un array vacío, no se genera ningún error.

***Nota**: los resultados devueltos por este comando pueden diferir en función del valor actual de la propiedad de la base "Considerar únicamente caracteres no alfanuméricos para las palabras claves" (ver el párrafo **Página Base de datos/Almacenamiento de datos**).*

GET PICTURE METADATA

GET PICTURE METADATA (imagen ; nomMeta ; ContenidoMeta {; nomMeta2 ; ContenidoMeta2 ; ... ; nomMetaN ; ContenidoMetaN})

Parámetro	Tipo		Descripción
imagen	Imagen	➡	Imagen de la cual obtener los metadatos
nomMeta	Texto	➡	Nombre o ruta de acceso del bloque a leer
ContenidoMeta	Variable	⬅	Contenido del metadato

Descripción

*El comando **GET PICTURE METADATA** permite leer el contenido de los metadatos (o meta-tags) presentes en imagen (campo o variable imagen 4D). Para mayor información sobre metadatos, consulte la descripción del comando **SET PICTURE METADATA**.*

En el parámetro nomMeta, pase una cadena especificando el tipo de metadato a recuperar. Puede pasar:

- *una constante del tema **Nombres de metadatos imágenes** con una ruta de etiqueta,*
- *el nombre de un bloque completo de metadatos ("TIFF", "EXIF", "GPS" o "IPTC"),*
- *una cadena vacía ("").*

Pase en el parámetro ContenidoMeta la variable destinada a recibir los metadatos.

- *Si pasa una ruta de etiqueta en nomMeta, el parámetro ContenidoMeta contiene directamente el valor a leer. El valor se convertirá en el tipo de la variable. Las variables de tipo texto serán formateadas en XML (estándar XMP). Puede pasar un array cuando el metadato contiene más de un valor (este es el caso, particularmente, para las etiquetas IPTC Keywords).*
- *Si pasa un nombre de bloque o una cadena vacía en nomMeta, el parámetro ContenidoMeta debe ser una referencia valida del elemento DOM XML. En este caso, el contenido del bloque designado (o de todos los bloques si pasó una cadena vacía en nomMeta) se copia nuevamente en el elemento de referencia.*

Ejemplo 1

Uso de estructuras de árbol DOM

```
$xml:=DOM Create XML Ref("Root") //Creación de un árbol XML DOM

 //Recepción de los metadatos TIFF
$_Xml_TIFF:=DOM Create XML element($xml;"/Root/TIFF")
GET PICTURE METADATA(vImagen;"TIFF";$_Xml_TIFF)

 //Recepción de los metadatos GPS
$_Xml_GPS:=DOM Create XML element($xml;"/Root/GPS")
GET PICTURE METADATA(vImagen;"GPS";$_Xml_GPS)
```

Ejemplo 2

Uso de variables

```
C_DATE($fechaComoFecha)
GET PICTURE METADATA(vImagen;TIFF DateTime;$fechaComoFecha)
 //devuelve únicamente la fecha ya que $fechaComoFecha es de tipo Fecha

C_TEXT($fechaComoTexto)
GET PICTURE METADATA(vImagen;TIFF/DateTime;$fechaComoTexto)
 //devuelve únicamente la fecha en formato XML

C_INTEGER($urgency)
GET PICTURE METADATA(vImagen;IPTC urgency;$urgencia)
```

Ejemplo 3

Recepción de etiquetas con valores múltiples en arrays

```
ARRAY TEXT($tTkeywords;0)
GET PICTURE METADATA(vImagen;IPTC keywords;$tTkeywords)
```

Después de la ejecución del comando, arrTkeywords contiene por ejemplo:

```
$arrTkeywords{1}="Francia"
$arrTkeywords{2}="Europa"
```

Ejemplo 4

Recepción de etiquetas con valores múltiples en una variable Texto

```
C_TEXT($vTwords;0)
GET PICTURE METADATA(vImagen;IPTC keywords;$vTwords)
```

Después de la ejecución del comando, vTwords contiene por ejemplo "Francia;Europa".

Variables y conjuntos del sistema

La variable sistema OK devuelve 1 si la recuperación de los metadatos es correcta y 0 si se produce un error o si no se encuentra al menos una de las etiquetas. En todos los casos, se devuelven los valores legibles.

Is picture file

Is picture file (rutaArchivo {; *}) -> Resultado

Parámetro	*Tipo*		*Descripción*
rutaArchivo	*Texto*	➡	*Ruta de acceso del archivo*
*	*Operador*	➡	*Validar los datos*
Resultado	*Booleano*	⮌	*True = rutaArchivo designa un archivo imagen, de lo contrario False*

Descripción

*El comando **Is picture fil**e prueba el archivo designado por el parámetro rutaArchivo y devuelve True si es un archivo de imagen válido. El comando devuelve False si el archivo no es de tipo imagen o si no se encuentra.*

Pase en el parámetro rutaArchivo la ruta de acceso del archivo imagen a probar. La ruta debe expresarse con la sintaxis del sistema. Puede pasar una ruta de acceso absoluta o relativa al archivo de estructura de la base. Si pasa una cadena vacía (""), el comando devuelve False.

*Si no pasa el parámetro *, el comando prueba el archivo buscando su extensión en la lista de codecs disponibles. Si quiere probar los archivos sin extensiones o efectuar una verificación más exhaustiva, pase el parámetro *. En este caso, el comando hace pruebas adicionales: carga e inspecciona el encabezado del archivo e interroga los codecs con el fin de validar la imagen. Esta sintaxis ralentiza la ejecución de comandos.*

***Nota**: el comando devuelve True para los archivos PDF en Windows y archivos EMF bajo Mac OS.*

PICTURE CODEC LIST

PICTURE CODEC LIST (arrayCodec {; arrayNoms}{; *})

Parámetro	Tipo		Descripción
arrayCodec	Array cadena	⇐	Identificadores de codecs de imágenes disponibles
arrayNoms	Array cadena	⇐	Nombres de los codecs de imágenes
*	Operador	⇒	Devuelve la lista de los codecs de lectura

Descripción

*El comando **PICTURE CODEC LIST** llena el array arrayCodec con la lista de los identificadores de los codecs de imágenes que están disponibles en el equipo donde se ejecuta. Esta lista incluye los codecs de los formatos de imágenes que son gestionados nativamente por 4D.*

Los identificadores de los codecs pueden devolverse en el array arrayCodec de las siguientes formas:

- *como una extensión (por ejemplo, ".gif")*
- *como un tipo Mime (por ejemplo, "Imagen/jpeg")*

***Nota de compatibilidad**: si QuickTime se ha activado en la base (ver la sección **Introducción a las imágenes**), los códigos QuickTime de 4 caracteres también pueden ser devueltos (por ejemplo "PNTG").*

La forma devuelta por el comando depende del modo de declaración del codec al nivel del sistema operativo. El array opcional arrayNoms permite recuperar el nombre de cada codec. Estos nombres son más explícitos que los identificadores. Este array puede utilizarse, por ejemplo, para crear y mostrar un menú que liste los codecs disponibles.

*Por defecto, si no pasa el parámetro *, el comando devuelve únicamente los codecs que pueden ser utilizados para codificar (escribir) las imágenes. Estos identificadores pueden utilizarse en el parámetro formato de los comandos de exportación de imágenes **WRITE PICTURE FILE** y **PICTURE TO BLOB**.*
*Si pasa el parámetro *, el comando también devuelve la lista de codecs utilizados para decodificar (leer) las imágenes. Las dos listas no son exclusivas, ciertos codecs de lectura y de escritura son idénticos. Los codecs destinados a la codificación de las imágenes pueden utilizarse para la decodificación. Por otra parte, los codecs de decodificación no necesariamente pueden utilizarse para la codificación. Por ejemplo, el codec ".jpg" se encontrará en ambas listas, mientras el ".xbmp" estará presente en la lista de codecs de lectura pero no en la de escritura.*

PICTURE LIBRARY LIST

PICTURE LIBRARY LIST (refsImag ; nomsImag)

Parámetro	Tipo		Descripción
refsImag	Array entero largo	⇐	Números de referencia de las imágenes de la librería de imágenes
nomsImag	Array cadena	⇐	Nombres de las imágenes de la librería de imágenes

Descripción

*El comando **PICTURE LIBRARY LIST** devuelve los números de referencia y los nombres de las imágenes almacenadas en la librería de imágenes de la base de datos.*

Después de llamarlo, usted recupera los números de referencia en el array refsImag y los nombres en el array nomsImag. Los dos arrays están sincronizados: el elemento n de refsImag es el número de referencia de la imagen de la librería cuyo nombre es devuelto en el elemento n de nomsImagenes.

Si es necesario, el comando crea y dimensiona automáticamente los arrays refsImagenes y nomsImagenes.

La longitud máxima del nombre de una imagen de la librería es de 255 caracteres.

Si la librería de imágenes está vacía, los dos arrays devueltos estarán vacíos.

*Para obtener el número de imágenes almacenadas actualmente en la librería de imágenes, utilice el comando **Size of Array** para obtener el tamaño de uno de los dos arrays.*

Ejemplo 1

El siguiente código devuelve el catálogo de la librería de imágenes en los arrays alRefImag y asNomImag:

```
PICTURE LIBRARY LIST(alRefImag;asNomImag)
```

Ejemplo 2

El siguiente ejemplo prueba si la librería de imágenes está vacía o no:

```
PICTURE LIBRARY LIST(alRefImag;asNomImag)
If(Size of array(alRefImag)=0)
    ALERT("La librería de imágenes está vacía.")
Else
    ALERT("La librería de imágenes contiene "+String(Size of array(alRefImag))+" imágenes.")
End if
```

Ejemplo 3

El siguiente ejemplo exporta la librería de imágenes a un documento almacenado en el disco:

```
PICTURE LIBRARY LIST($alPicRef;$asPicName)
$vlNbPictures:=Size of array($alPicRef)
If($vlNbPictures>0)
    SET CHANNEL(12;"")
    If(OK=1)
        $vsTag:="4DV6PICTURELIBRARYEXPORT"
        SEND VARIABLE($vsTag)
        SEND VARIABLE($vlNbPictures)
        gError:=0
        For($vlPicture;1;$vlNbPictures)
            $vlPicRef:=$alPicRef{$vlPicture}
            $vsPicName:=$asPicName{$vlPicture}
            GET PICTURE FROM LIBRARY($alPicRef{$vlPicture};$vgPicture)
            If(OK=1)
                SEND VARIABLE($vlPicRef)
                SEND VARIABLE($vsPicName)
                SEND VARIABLE($vgPicture)
            Else
                $vlPicture:=$vlPicture+1
                gError:=-108
            End if
        End for
        SET CHANNEL(11)
        If(gError#0)
```

```
            ALERT("La librería de imágenes no pudo exportarse, inténtelo con más memoria.")
            DELETE DOCUMENT(Document)
        End if
    End if
Else
    ALERT("La librería de imágenes está vacía.")
End if
```

PICTURE PROPERTIES

PICTURE PROPERTIES (imagen ; largo ; altura {; hDesp {; vDesp {; modo}}})

Parámetro	*Tipo*		*Descripción*
imagen	*Imagen*	➡	*Imagen para la cual obtener la información*
largo	*Entero largo*	⬅	*Largo de la imagen expresado en píxeles*
altura	*Entero largo*	⬅	*Alto de la imagen expresado en píxeles*
hDesp	*Entero largo*	⬅	*Offset horizontal cuando la imagen se muestra en segundo plano*
vDesp	*Entero largo*	⬅	*Offset vertical cuando la imagen se muestra en segundo plano*
modo	*Entero largo*	⬅	*Modo de transferencia cuando la imagen se muestra en segundo plano*

Descripción

*El comando **PICTURE PROPERTIES** devuelve la información sobre la imagen que pasa en el parámetro imagen.*

Los parámetros largo y alto devuelven el largo y alto de la imagen.

Los parámetros hDesp, vDesp, y modo devuelven las posiciones horizontal y vertical y el modo de transferencia de la imagen cuando se muestra en el fondo en un formulario ("Imagen de fondo").

Picture size

Picture size (imagen) -> Resultado

Parámetro	*Tipo*		*Descripción*
imagen	*Imagen*	→	*Imagen para la cual devolver el tamaño en bytes*
Resultado	*Entero largo*	↩	*Tamaño en bytes de la imagen*

Descripción

Esta función devuelve el tamaño de imagen en bytes.

PICTURE TO BLOB

PICTURE TO BLOB (imagen ; blobImag ; codec)

Parámetro	Tipo		Descripción
imagen	Imagen	➡	Campo o variable tipo imagen
blobImag	BLOB	⬅	BLOB para recibir la imagen convertida
codec	Cadena	➡	Identificación de codec de imagen

Descripción

*El comando **PICTURE TO BLOB** convierte una imagen almacenada en una variable o en un campo 4D en otro formato y ubica la imagen resultante en un BLOB.*

Usted pasa en el parámetro imagen una variable o un campo 4D de tipo imagen y en el parámetro blobImag la variable o el campo BLOB el cual debe contener la imagen convertida.

Pase en el parámetro codec una cadena indicando el formato de conversión.

Un Codec puede ser una extensión (por ejemplo, "gif") o un tipo Mime (por ejemplo "image/jpg).
*Puede obtener la lista de códigos disponibles vía el comando **PICTURE CODEC LIST.***

Una vez el comando ha sido ejecutado, blobImagen contiene la imagen en el formato especificado.

Si la conversión fue exitosa, la variable sistema OK toma el valor 1. Si la conversión falla (convertidor no disponible), OK toma el valor 0 y el BLOB se genera vacío (0 byte).

Ejemplo

Desea convertir una imagen de un formato propietario a formato GIF y mostrarlo en una página web estática. Puede utilizar un código como:

```
C_PICTURE($picture)
C_BLOB($BLOB)
C_TEXT($path)

$path:=Get 4D folder(Current resources folder)+"Images"+Folder separator+"Sunrise.psd" //encontrar la imagen en la carpeta Imágenes de la carpeta Recursos
READ PICTURE FILE($path;$picture) //poner la imagen en la variable imagen

PICTURE TO BLOB($picture;$BLOB;".gif") //convertir la imagen al formato ".gif"
WEB SEND BLOB($BLOB;"image/gif")
```

READ PICTURE FILE

READ PICTURE FILE (nomArchivo ; imagen {; *})

Parámetro	Tipo		Descripción
nomArchivo	Cadena	➡	Nombre o ruta de acceso completa del archivo a leer, o cadena vacía
imagen	Imagen	⬅	Campo o variable que recibe la imagen
*	Operador	➡	Si se pasa = acepta todo tipo de archivo

Descripción

El comando ***READ PICTURE FILE*** *permite abrir la imagen guardada en el archivo del disco nomArchivo y cargarla en el campo o variable 4D imagen.*

Puede pasar en nomArchivo la ruta de acceso completa del archivo a leer, o únicamente el nombre del archivo. Si solo pasa el nombre del archivo, el archivo será ubicado junto al archivo de estructura de la base. Bajo Windows, igualmente debe indicar la extensión del archivo.

Si pasa una cadena vacía ("") en nomArchivo, aparece la caja de diálogo estándar de apertura de archivos, permitiendo al usuario seleccionar el archivo a leer, así como los formatos disponibles.

Puede obtener la lista de formatos disponibles con la ayuda del comando ***PICTURE CODEC LIST****.*

Pase en imagen la variable o el campo imagen que deba recibir la imagen leída.

Nota: *el formato interno de la imagen se almacena dentro de la variable o campo 4D.*

*Si pasa el parámetro opcional *, el comando aceptará todo tipo de archivo. Esto significa que puede trabajar con imágenes sin necesariamente disponer de los codecs adecuados (ver la descripción del comando* ***BLOB TO PICTURE****).*

Variables y conjuntos del sistema

Si la ejecución del comando es correcta, la variable sistema Document contiene la ruta de acceso completa al archivo abierto y la variable sistema OK toma el valor 1. De lo contrario, OK toma el valor 0.

REMOVE PICTURE FROM LIBRARY

REMOVE PICTURE FROM LIBRARY (refImag | nomImag)

Parámetro	Tipo		Descripción
refImag | nomImag	Entero largo, Cadena	➡	Número de referencia o nombre de una imagen de la librería de imágenes

Descripción

*El comando **REMOVE PICTURE FROM LIBRARY** elimina de la librería de imágenes la imagen cuyo número de referencia se pasa en refImag o cuyo nombre se pasa en nomImag.*

Si el número de referencia o nombre no corresponden a ninguna imagen, el comando no hace nada.

***4D Server: REMOVE PICTURE FROM LIBRARY** no puede utilizarse desde un método ejecutado en el equipo servidor (procedimiento almacenado o trigger). Si llama **REMOVE PICTURE FROM LIBRARY** en un equipo servidor, no pasa nada, se ignora la llamada.*

***Advertencia:** los objetos de diseño (elementos de lista jerárquica, líneas de menú, etc.) pueden hacer referencia a una imagen de la librería. Sea prudente cuando elimine por programación una imagen de la librería de imágenes.*

Ejemplo 1

El siguiente ejemplo borra la imagen #4444 de la librería de imágenes.

```
REMOVE PICTURE FROM LIBRARY(4444)
```

Ejemplo 2

El siguiente ejemplo borra de la librería de imágenes toda imagen cuyo nombre comience por el símbolo dólar ($):

```
PICTURE LIBRARY LIST($alRefImag;$asNomImag)
For($vlImag;1;Size of array($alRefImag))
   If($asNomImag{$vlImag}="$@")
      REMOVE PICTURE FROM LIBRARY($alRefImag{$vlImag})
   End if
End for
```

SET PICTURE FILE NAME

SET PICTURE FILE NAME (imagen ; nomArchivo)

Parámetro	Tipo		Descripción
imagen	Campo imagen, Variable imagen	⇒	Imagen para la cual definir el nombre por defecto
nomArchivo	Texto	⇒	Nombre de la imagen por defecto

Descripción

*El comando **SET PICTURE FILE NAME** define o modifica el nombre del archivo por defecto de la imagen pasada como parámetro.*

*Este nombre se puede definir automáticamente a partir del nombre de origen del archivo imagen importado en el campo o variable imagen o durante una llamada previa a **SET PICTURE FILE NAME**.*

El nombre por defecto se utiliza como nombre de archivo cuando la imagen se exporta en un archivo disco. Si el contenido del campo se copia en una variable o en otro campo, el nombre por defecto también se copia. Para mayor información, consulte el Manual de Diseño.

SET PICTURE METADATA

SET PICTURE METADATA (imagen ; nomMeta ; ContenidoMeta {; nomMeta2 ; ContenidoMeta2 ; ... ; nomMetaN ; ContenidoMetaN})

Parámetro	Tipo		Descripción
imagen	Imagen	➡	Imagen cuyos metadatos quiere escribir
nomMeta	Texto	➡	Nombre o ruta del bloque a escribir
ContenidoMeta	Variable	➡	Contenido del metadato

Descripción

*El comando **SET PICTURE METADATA** permite escribir o modificar el contenido de los metadatos (o meta-etiquetas) presentes en imagen (campo o una variable imagen 4D), cuando son modificables.*

Los metadatos son información adicional insertada en las imágenes. 4D permite manipular cuatro tipos de metadatos estándar: EXIF, GPS, IPTC y TIFF.

***Nota:** para una descripción detallada de estos tipos de metadatos, puede consultar los siguientes documentos: http://www.iptc.org/std/IIM/4.1/specification/IIMV4.1.pdf (IPTC) y http://exif.org/Exif2-2.PDF (TIFF, EXIF y GPS).*

En el parámetro nomMeta, pase una cadena especificando el tipo de metadato a escribir o modificar. Puede pasar:

- *una de las constantes del tema **Nombres de metadatos imágenes**. Este tema agrupa todas las etiquetas soportadas por 4D. Cada constante contiene una ruta de etiqueta (por ejemplo, "TIFF/DateTime"),*
- *el nombre de un bloque completo de metadatos ("TIFF", "EXIF", "GPS" o "IPTC"),*
- *una cadena vacía ("").*

Pase los nuevos valores del metadato en el parámetro contenidoMeta:

- *Si pasa una constante de ruta de etiqueta en nomMeta, pase directamente en contenidoMeta el valor a escribir o una de las constantes del tema **Picture Metadata Values**. El valor puede ser de tipo Texto, Entero largo, Real, Fecha u Hora, de acuerdo al metadato especificado. Puede utilizar un array si el metadato contiene más de un valor. Si pasa una cadena, debe ser formateada en XML (estándar XMP). Puede pasar una cadena vacía ("") para borrar todo metadato existente.*
- *Si pasa un nombre de bloque o una cadena vacía en nomMeta, pase en contenidoMeta la referencia XML DOM del elemento que contiene los metadatos a escribir. En el caso de una cadena vacía, todos los metadatos se modificarán.*

***Atención:** ciertos metadatos están en modo sólo lectura y por lo tanto no pueden ser modificados por el comando **SET PICTURE METADATA**, por ejemplo TIFF XResolution/TIFF YResolution, EXIF Color Space or EXIF Pixel X Dimension/EXIF Pixel Y Dimension.*
Bajo Windows, si ocurre un error durante la ejecución del comando, la variable OK toma el valor 0. Note que bajo Mac OS, por razones técnicas, los errores de escritura de metadatos no se detectan. Por lo tanto este comando no modifica la variable OK bajo MacOS.

Notas:

- *Sólo ciertos formatos de imágenes (específicamente JPEG y TIFF) soportan los metadatos. Por el contrario, los formatos tales como GIF o BMP no aceptan los metadatos. Cuando convierte una imagen con metadatos a un formato que no los soporta, se pierde la información.*
- *Bajo OS X versión 10.7 (Lion), un bug del framework nativo utilizado para la codificación y decodificación de los metadatos puede causar errores de precisión en las coordenadas de GPS. En este caso, se recomienda una actualización a OS X 10.8 (Mountain Lion) o 10.9 (Maverick).*

Ejemplo 1

Escritura de varios valores del metadato "Keywords" por medio de arrays:

```
ARRAY TEXT($arrTkeywords;2)
$arrTkeywords{1}:="Francia"
$arrTkeywords{2}:="Europa"
SET PICTURE METADATA(vPicture;IPTC keywords;$arrTkeywords)
```

Ejemplo 2

Escritura del bloque GPS vía una referencia DOM:

```
C_TEXT($domMetas)
$domMetas:=DOM Parse XML source("metas.xml")
C_TEXT($gpsRef)
$gpsRef:=DOM Find XML element($domMetas;"Metadatas/GPS")
If(OK=1)
   SET PICTURE METADATA(vImage;"GPS";$refGPS)
 //$gpsRef realmente apunta al elemento GPS
   ...
```

```
End if
DOM CLOSE XML($domMetas)
```

Nota

Cuanto todos los metadatos se manipulan vía una referencia de elementos DOM, las etiquetas se guardan como atributos asociados a un elemento (hijo del elemento referenciado) cuyo nombre es el nombre del bloque (TIFF, IPTC, etc.). Cuando se manipula un bloque de metadatos específico, las etiquetas del bloque se almacenan como atributos directamente asociados al elemento referenciado por el comando.

SET PICTURE TO LIBRARY

SET PICTURE TO LIBRARY (imagen ; refImag ; nomImag)

Parámetro	Tipo		Descripción
imagen	Imagen	➡	Nueva imagen
refImag	Entero largo	➡	Número de referencia de la imagen en la librería de imágenes
nomImag	Cadena	➡	Nuevo nombre de la imagen

Descripción

*El comando **SET PICTURE TO LIBRARY** crea una nueva imagen o reemplaza una imagen existente en la librería de imágenes. Antes de llamar el comando, usted pasa:*

- *el número de referencia de la imagen en refImag (entre 1 y 32767)*
- *la imagen misma en imagen.*
- *el nombre de la imagen en nomImag (longitud máxima: 255 caracteres).*

Si hay una imagen en la librería de imágenes con el mismo número de referencia, su contenido será reemplazado y la imagen se renombra con los valores pasados en imagen y nomImag.

Si no hay una imagen en la librería de imágenes con el número de referencia pasado en refImag, una nueva imagen se añade a la librería de imágenes.

***4D Server: SET PICTURE TO LIBRARY** no puede utilizarse dentro de un método ejecutado en el equipo servidor (procedimiento almacenado o trigger). Si llama **SET PICTURE TO LIBRARY** en un equipo servidor, no pasa nada, la llamada se ignora.*

***Advertencia:** los objetos de estructura (elementos de listas jerárquicas, líneas de menú, etc.) pueden referirse a una imagen de la librería de imágenes. Sea prudente cuando modifique por programación una imagen de la librería de imágenes.*

***Nota:** si pasa una imagen vacía en imagen o un valor negativo o nulo en refImag, el comando no hace nada.*

Ejemplo 1

Sin importar el contenido actual de la librería de imágenes, el siguiente ejemplo añade una nueva imagen a la librería buscando primero un número de referencia de una imagen única:

```
PICTURE LIBRARY LIST($alRefImag;$asNomsImag)
Repeat
   $vlRefImag:=1+Abs(Random)
Until(Find in array($alRefImag;$vlRefImag)<0)
SET PICTURE TO LIBRARY(vgImagen;$vlRefImag;"Nueva Imagen")
```

Ejemplo 2

*El siguiente ejemplo importa en la librería de imágenes las imágenes (almacenadas en un documento en disco) creadas por el tercer ejemplo del comando **PICTURE LIBRARY LIST**:*

```
SET CHANNEL(10;"")
If(OK=1)
   RECEIVE VARIABLE($vsTag)
   If($vsTag="4DV6PICTURELIBRARYEXPORT")
      RECEIVE VARIABLE($vlNbImagenes)
      If($vlNbImagenes>0)
         For($vlImag;1;$vlNbImagenes)
            RECEIVE VARIABLE($vlPicRef)
            If(OK=1)
               RECEIVE VARIABLE($vsNomImag)
            End if
            If(OK=1)
               RECEIVE VARIABLE($vgImag)
            End if
            If(OK=1)
               SET PICTURE TO LIBRARY($vgImag;$vlRefImag;$vsNomImag)
            Else
               $vlImag:=$vlNbImagenes+1
               ALERT("Este archivo parece estar dañado.")
            End if
         End for
      Else
         ALERT("Este archivo parece estar dañado.")
```

```
        End if
    Else
        ALERT("El archivo ""+Document+"" no es un archivo de exportación de la librería de imágenes.")
    End if
    SET CHANNEL(11)
End if
```

Gestión de errores

Si no hay suficiente memoria para añadir la imagen a la librería de imágenes, se genera un error -108. Note que los errores E/S también pueden ser generados (si por ejemplo el archivo de estructura está bloqueado). Puede interceptar estos errores con un método de gestión de errores.

TRANSFORM PICTURE

TRANSFORM PICTURE (imagen ; operador {; param1 {; param2 {; param3 {; param4}}}})

Parámetro	Tipo		Descripción
imagen	Imagen	➡	Imagen fuente a transformar
		⬅	Imagen resultante de la transformación
operador	Entero largo	➡	Tipo de transformación a efectuar
param1	Real	➡	Parámetro de la transformación
param2	Real	➡	Parámetro de la transformación
param3	Real	➡	Parámetro de la transformación
param4	Real	➡	Parámetro de la transformación

Descripción

*El comando **TRANSFORM PICTURE** permite aplicar una transformación de tipo operador a la imagen pasada en el parámetro imagen.*

***Nota:** este comando extiende las funcionalidades ofrecidas por los operadores convencionales de transformación de imágenes (+/, etc., ver la sección **Operadores de imágenes**). Estos operadores permanecen totalmente utilizables en 4D.*

La imagen fuente se modifica directamente después de la ejecución del comando. Tenga en cuenta que ciertas operaciones no son destructivas y pueden revertirse mediante la realización de la operación contraria o por medio de la operación "Reset". Por ejemplo, una imagen reducida a 1% retomará su tamaño original sin alteraciones si se agranda 100 veces. Las transformaciones no modifican el tipo original de la imagen: por ejemplo, una imagen vectorial permanecerá vectorial después de su transformación.

*En operador, pase el número de la operación a efectuar y en param1 a param4, el o los parámetro(s) necesarios para esta operación (el número de parámetros depende de la operación). En operador puede utilizar una de las constantes del tema "**Transformación de imágenes**". Estos operadores y sus parámetros se describen en la siguiente tabla:*

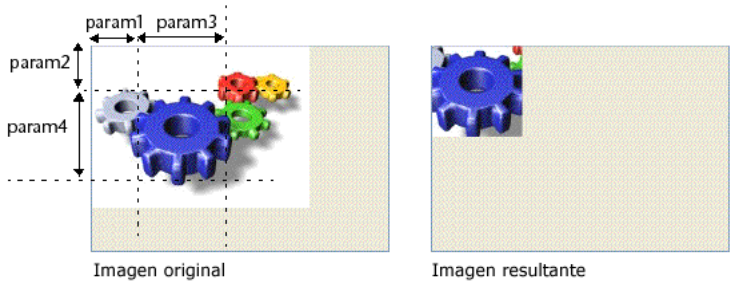
operador (valor)	param1	param2	param3	param4	Valores	Cancelable
Reset (0)	-	-	-	-	-	-
Scale (1)	Ancho	Alto	-	-	Factores	Sí
Translate (2)	Eje X	Eje Y	-	-	Pixeles	Sí
Flip horizontally (3)	-	-	-	-		Sí
Flip vertically (4)	-	-	-	-		Sí
Crop (100)	X Orig.	Y Orig.	Ancho	Alto	Pixeles	No
Fade to grey scale (101)	-	-	-	-		No
Transparency (102)	RGB color	-	-	-	Hexadécimal	No

- *Reset: todas las operaciones matriciales efectuadas en la imagen (redimensionar, voltear, etc.) se deshacen.*
- *Scale: la imagen se redimensiona horizontalmente y verticalmente de acuerdo a los valores pasados en param1 y param2 respectivamente. Estos valores son factores: por ejemplo, para agrandar el ancho 50%, pase 1.5 en param1 y para reducir la altura 50%, pase 0.5 en param2.*
- *Translate: la imagen se mueve param1 píxeles horizontalmente y param2 píxeles verticalmente. Pase un valor positivo para moverse a la derecha o hacia abajo y un valor negativo para moverse hacia la izquierda o hacia arriba.*
- *Flip horizontally y Flip vertically: el efecto espejo se aplica a la imagen original. Todo movimiento efectuado anteriormente no se tendrá en cuenta.*
- *Crop: la imagen se recorta a partir del punto de coordenadas param1 y param2 (expresado en píxeles). El ancho y el alto de la nueva imagen son determinados por los parámetros param3 y param4. Esta transformación no puede anularse.*
- *Fade to grey scale: la imagen se convierte a escala de grises (ningún parámetro es necesario). Esta transformación no puede deshacerse.*
- *Transparency: una máscara de transparencia se aplica a la imagen basada en el color pasado en param1. Por ejemplo, si pasa 0x00FFFFFF (color blanco) en param1, todos los píxeles blancos en la imagen original serán transparentes en la imagen transformada. Esta operación se puede aplicar a imágenes de mapa de bits o de vectores. Por defecto, si el parámetro param1 se omite, el color blanco (0x00FFFFFF) se establece como color objetivo. Esta función está especialmente diseñada para manejar la transparencia en imágenes convertidas desde el formato obsoleto PICT, pero se puede usar con imágenes de todo tipo. Esta transformación no se puede deshacer.*

Ejemplo 1

Este es un ejemplo de corte de una imagen (la imagen se muestra en el formulario con el formato "Truncado (no-centrado")):

```
TRANSFORM PICTURE($vpEngranaje;Crop;50;50;100;100)
```

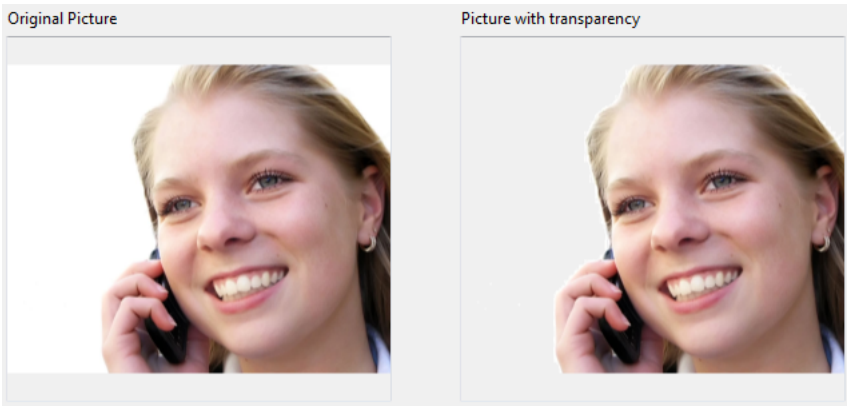


Ejemplo 2

Quiere definir las partes blancas de una imagen como transparentes. Para ello, puede utilizar el siguiente código:

```
TRANSFORM PICTURE(Pict1;Transparency;0x00FFFFFF) //0x00FFFFFF es blanco
```

Obtiene el siguiente resultado:



WRITE PICTURE FILE

WRITE PICTURE FILE (nomArchivo ; imagen {; codec})

Parámetro	Tipo		Descripción
nomArchivo	Alpha	➡	Nombre o ruta de acceso completo del archivo a escribir, o cadena vacía
imagen	Imagen	➡	Campo o variable imagen a escribir
codec	Cadena	➡	Identificación de codec de imagen

Descripción

*El comando **WRITE PICTURE FILE** permite guardar en un archivo en el disco la imagen pasada en el parámetro imagen, en el formato definido por codec.*

Puede pasar en nomArchivo la ruta de acceso completa del archivo a crear, o únicamente el nombre del archivo. Si solo pasa el nombre del archivo, el archivo será ubicado junto al archivo de estructura de la base.

Tiene que indicarse la extensión del archivo.

*Si pasa una cadena vacía ("") en nomArchivo, aparece la caja de diálogo estándar de registro, permitiendo al usuario indicar el nombre, ubicación y formato del archivo a crear. Si se ha asociado un nombre por defecto al campo Imagen, se suministra en la caja de diálogo (ver el comando **SET PICTURE FILE NAME**).*

Pase en imagen la variable o campo imagen que contiene la imagen a almacenar en el disco.

El parámetro opcional codec permite definir el formato en el cual la imagen se guardará. Un codec puede ser una extensión (por ejemplo ".gif") o un tipo Mime (por ejemplo "image/jpg").
*Puede obtener una lista de codecs disponibles a través del comando **PICTURE CODEC LIST**.*

Si se omite el parámetro codec, el comando intentará determinar el codec basado en la extensión del nombre del archivo pasado en el parámetro nomArchivo. Por ejemplo, si pasa la instrucción:

```
WRITE PICTURE FILE("c:\folder\photo.jpg";myphoto)
```

... el comando utilizará el codec JPEG para guardar la imagen.
Si la extensión utilizada no corresponde a ningún codec disponible, el archivo no se guarda y la variable sistema OK toma el valor 0. Si no pasa un codec o un archivo de extensión, el archivo imagen se guarda en formato PICT.

***Nota:** si el formato de escritura de imagen (indicado vía la extensión del nomArchivo o el parámetro codec) es igual a su tipo original y si ninguna operación de transformación se ha aplicado, la imagen se escribe "tal cual", sin ninguna modificación.*

Si la ejecución del comando es correcta, la variable sistema Document contiene la ruta de acceso completa al archivo creado y la variable sistema OK toma el valor 1. De lo contrario, OK toma el valor 0.

_o_PICTURE TO GIF

_o_PICTURE TO GIF (imag ; blobGIF)

Parámetro	Tipo		Descripción
imag	Imagen	➡	Campo o variable imagen
blobGIF	BLOB	⬅	BLOB que contiene la imagen de tipo GIF

Nota de compatibilidad

*Este comando está obsoleto. Puede utilizar el comando **PICTURE TO BLOB** para realizar la misma conversión.*

_o_PICTURE TYPE LIST

_o_PICTURE TYPE LIST (arrayFormatos {; arrNombres})

Parámetro	Tipo		Descripción
arrayFormatos	*Array cadena*	⇐	*Códigos QuickTime de los formatos de importación/exportación disponibles*
arrNombres	*Array cadena*	⇐	*Nombres de los formatos*

Nota de compatibilidad:

*Este comando requiere de QuickTime y no ofrece acceso a los formatos administrados de forma nativa por 4D desde la versión 11. En lo sucesivo su interés es limitado y puede ser reemplazado favorablemente por el comando **PICTURE CODEC LIST**.*

_o_QT COMPRESS PICTURE

_o_QT COMPRESS PICTURE (imagen ; metodo ; calidad)

Parámetro	Tipo		Descripción
imagen	Imagen	➡	Imagen a comprimir
		⬅	Imagen comprimida
metodo	Cadena	➡	Método de compresión cadena 4 caracteres
calidad	Entero largo	➡	Calidad de compresión (1..1000)

Nota de compatibilidad

*Este comando llama mecanismos obsoletos y debe ser reemplazado por el comando **CONVERT PICTURE**.*

_o_QT COMPRESS PICTURE FILE

_o_QT COMPRESS PICTURE FILE (documento ; metodo ; calidad)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Número de referencia del documento
metodo	Cadena	➡	Método de compresión cadena 4 caracteres
calidad	Entero largo	➡	Calidad de compresión (1..1000)

Nota de compatibilidad

*Este comando llama mecanismos obsoletos y debe ser reemplazado por los comandos **WRITE PICTURE FILE** o **PICTURE TO BLOB**.*

_o_QT LOAD COMPRESS PICTURE FROM FILE

_o_QT LOAD COMPRESS PICTURE FROM FILE (documento ; metodo ; calidad ; imagen)

Parámetro	Tipo		Descripción
documento	DocRef	⇒	Número de referencia del documento
metodo	Cadena	⇒	Método de compresión cadena 4 caracteres
calidad	Entero largo	⇒	Calidad de compresión (1..1000)
imagen	Imagen	⇐	Imagen comprimida

Nota de compatibilidad

*Este comando llama mecanismos obsoletos y debe ser reemplazado por los comandos **READ PICTURE FILE** y **CONVERT PICTURE**.*

_o_SAVE PICTURE TO FILE

_o_SAVE PICTURE TO FILE (documento ; imagen)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Número de referencia del documento
imagen	Imagen	➡	Imagen a guardar

Nota de compatibilidad:

Este comando llama mecanismos obsoletos y sólo se mantiene por razones de compatibilidad. Ha sido reemplazado favorablemente por el comando ***WRITE PICTURE FILE.***

Importación y exportación

- *EXPORT DATA*
- *EXPORT DIF*
- *EXPORT ODBC*
- *EXPORT SYLK*
- *EXPORT TEXT*
- *IMPORT DATA*
- *IMPORT DIF*
- *IMPORT ODBC*
- *IMPORT SYLK*
- *IMPORT TEXT*

EXPORT DATA

EXPORT DATA (nomArchivo {; proyecto {; *}})

Parámetro	Tipo		Descripción
nomArchivo	Cadena	➡	Ruta de acceso y nombre del archivo a exportar
proyecto	Variable texto, BLOB variable	➡	Contenido del proyecto de exportación
		⬅	Nuevo contenido del proyecto de exportación (si se pasa el parámetro *)
*	Operador	➡	Visualización de la caja de diálogo de exportación y actualización del proyecto

Descripción

*El comando **EXPORT DATA** exportar datos en el archivo nomArchivo. 4D puede exportar datos en los siguientes formatos: Texto, Texto de longitud fija, XML, SYLK, DIF, DBF (dBase) y 4D.*

*Si pasa una cadena vacía en nomArchivo, **EXPORT DATA** muestra la caja de diálogo estándar de guardar archivos, permitiendo al usuario definir el nombre, tipo y ubicación del archivo de exportación. Una vez aceptada la caja de diálogo, la variable sistema **Documento** contiene la ruta de acceso y el nombre del archivo. Si el usuario hace clic en **Cancelar**, se detiene la ejecución del comando y la variable sistema **OK** toma el valor 0.*
*El parámetro opcional proyecto permite utilizar un proyecto para exportar datos. Cuando pasa este parámetro, se lleva a cabo la exportación directamente, sin ninguna intervención (a menos de que utilice la opción *, ver a continuación). Si no pasa este parámetro, aparece la caja de diálogo de exportación. El usuario puede definir sus parámetros de exportación o cargar un proyecto de exportación existente.*

Un proyecto de exportación contiene todos los parámetros de exportación, tales como las tablas y campos a exportar, los delimitadores, etc. En el parámetro proyecto puede pasar una variable Texto con XML o una variable Texto con una referencia a un elemento DOM pre existente, o un BLOB. Los proyectos pueden crearse por programación (proyectos de formato XML únicamente) o cargando los parámetros previamente definidos en la caja de diálogo de exportación. En el último caso, hay dos soluciones disponibles:

- *Usar el comando **EXPORT DATA** con un parámetro proyecto vacío y el parámetro opcional *, luego guardar el parámetro proyecto en un campo Texto o BLOB (ver a continuación). Esta solución le permite guardar el proyecto con el archivo de datos.*
- *Guardar el proyecto en el disco, luego cargarlo utilizando el comando **DOM Parse XML source** y pasando su referencia en el parámetro proyecto.*

***Nota de compatibilidad**: a partir de la versión 12 de 4D, los proyectos de exportación son codificados en XML. 4D puede abrir los proyectos de exportación generados con las versiones anteriores de 4D (formato BLOB), sin embargo los proyectos creados a partir de la v12 no pueden abrirse con una versión 11 o anterior. Además se recomienda utilizar variables Texto para manipular los archivos de exportación.*

*El parámetro opcional *, si se especifica, hace que aparezca la caja de diálogo de exportación con los parámetros definidos en proyecto. Esta característica le permite utilizar un proyecto predefinido, mientras tiene aún la posibilidad de modificar uno o más parámetros. Además, el parámetro proyecto contiene, después de cerrar la caja de diálogo de exportación, los parámetros del "nuevo" proyecto. Entonces puede almacenar el nuevo proyecto en un campo BLOB, en disco, etc.*

*Si la exportación fue exitosa, la variable sistema **OK** es igual a 1.*

Ejemplo 1

*Este ejemplo presenta el uso del comando **EXPORT DATA** para exportar datos en formato binario.*

- *Este método hace un bucle por todas las tablas de la base y llama al método **ExportBinary**:*

```
C_TEXT($ExportPath)
C_LONGINT($i)
$ExportPath:=Select folder("Por favor seleccione la carpeta de exportación:")
If(Ok=1)
   For($i;1;Get last table number)
      If(Is table number valid($i))
         ExportBinary(Table($i);$ExportPath+Table name($i);True)
      End if
   End for
End if
```

- *Este es el código del método **ExportBinary**:*

```
C_POINTER($1) //tabla
C_TEXT($2) //ruta del archivo de destino
C_BOOLEAN($3) //exportar todos los registros
C_LONGINT($i)
C_TEXT($ref)
$ref:=DOM Create XML Ref("settings-import-export")
 // Exportar la tabla "$1" en formato binario '4D', todos los registros o únicamente la selección actual
```

```
DOM SET XML ATTRIBUTE($ref;"table_no";Table($1);"format";"4D";"all_records";$3)
 // Definición de los campos a exportar
For($i;1;Get last field number($1))
    If(Is field number valid($1;$i))
        $elt:=DOM Create XML element($ref;"field";"table_no";Table($1);"field_no";$i)
    End if
End for
EXPORT DATA($2;$ref)
If(Ok=0)
    ALERT("Error durante la exportación de la tabla "+Table name($1))
End if
DOM CLOSE XML($ref)
```

Ejemplo 2

Este ejemplo crea un proyecto vacío y guarda los parámetros definidos por el usuario en la caja de diálogo de exportación:

```
C_TEXT($exportParams)
EXPORT DATA("DocExport.txt";$exportParams;*) ` Visualización de la caja de diálogo de exportación
```

Variables y conjuntos del sistema

*Si el usuario hace clic en **Cancelar** en la caja de diálogo estándar de abrir archivos o de exportación, la variable sistema OK toma el valor 0. Si la exportación fue exitosa, la variable sistema OK toma el valor 1.*

EXPORT DIF

EXPORT DIF ({tabla ;} doc)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual exportar datos, o Tabla por defecto, si se omite
doc	Cadena	➡	Documento DIF para recibir los datos

Descripción

El comando **EXPORT DIF** escribe los datos de los registros de la selección actual de tabla en el proceso actual. Los datos se escriben en documento, un documento DIF Windows o Macintosh, en el disco.

La operación de exportación se efectúa por el formulario de salida actual. La operación de exportación escribe los campos y las variables basado en el orden de entrada del formulario de salida. Por esta razón, utilice un formulario de salida que sólo contenga los campos u objetos editables que quiera exportar. No ubique botones u otros objetos en el formulario de exportación. Los objetos de subformulario se ignoran.

Un evento On Load se envía al método del formulario por cada registro exportado. Utilice este evento para definir las variables que podría utilizar en el formulario de exportación.

El parámetro documento puede dar nombre a un documento nuevo o existente. Si documento tiene el mismo nombre que un documento existente, el documento existente se sobrescribe. El documento puede incluir una ruta de acceso que contenga los nombres de los volúmenes y las carpetas. Si pasa una cadena vacía, aparece la caja de diálogo estándar de guardar archivos. Si el usuario cancela este diálogo, se cancela la operación de exportación, y la variable sistema OK toma el valor 0.

Un termómetro de progreso aparece durante la exportación. El usuario puede cancelar la operación haciendo clic en el botón Detener. Si la exportación termina con éxito, la variable sistema OK toma el valor 1. En caso de error o si la operación se interrumpe, la variable OK toma el valor 0. El termómetro puede ocultarse con el comando **MESSAGES OFF**.

El comando utiliza por defecto el conjunto de caracteres UTF-8. Los documento en formato DIF utilizan por lo general el conjunto de caracteres IBM437, puede utilizar el comando **USE CHARACTER SET** para definir el conjunto de caracteres apropiado.

Durante la utilización de **EXPORT DIF** , el delimitador de campos por defecto es el carácter de tabulación (código 9). El delimitador del registro por defecto es el retorno de carro (código 13). Puede modificar estos valores asignando nuevos valores a las dos **Variables sistema** FldDelimit y RecDelimit. El usuario puede cambiar estos valores en la caja de diálogo de exportación del entorno Diseño. Como los campos Texto pueden contener retornos de carro, sea cuidadoso si utiliza el retorno de carro como delimitador entre los campos a exportar.

Ejemplo

El siguiente ejemplo exporta los datos a un documento DIF. El método comienza por definir el formulario de salida de manera que los datos se exporten por el formulario correcto, luego realiza la exportación:

```
FORM SET OUTPUT([Personas];"Exportar")
EXPORT DIF([Personas];"Nuevas_Personas.dif") ` Exporta al documento "Nuevas_Personas.dif"
```

Variables y conjuntos del sistema

OK toma el valor 1 si la exportación termina con éxito; de lo contrario, toma el valor 0.

EXPORT ODBC

EXPORT ODBC (tablaFuente {; proyecto {; *}})

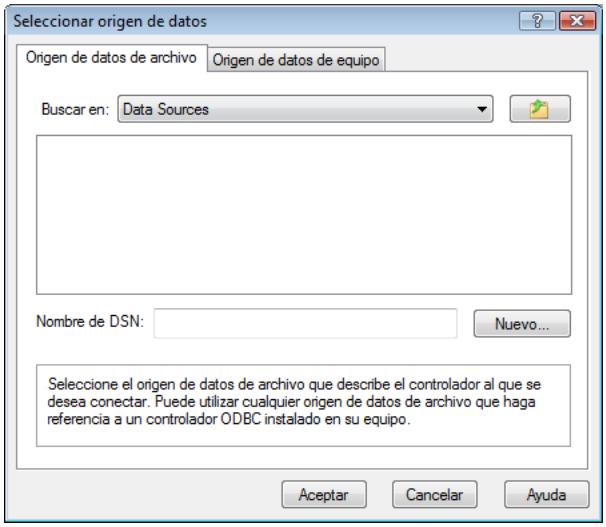
Parámetro	Tipo		Descripción
tablaFuente	Cadena	➡	Nombre de la tabla en la fuente de datos ODBC
proyecto	BLOB	➡	Contenido del proyecto de exportación
		⬅	Nuevo contenido del proyecto de exportación (si se pasa *)
*	Operador	➡	Muestra de la caja de diálogo de exportación y actualización del proyecto

Descripción

*El comando **EXPORT ODBC** permite exportar datos en la tabla tablaFuente de una fuente ODBC externa.*

*Si llama al comando **EXPORT ODBC** fuera de una conexión previamente abierta con la ayuda del comando , aparece la caja de diálogo de selección de la fuente de datos ODBC:*

Windows



Mac OS

*Si el usuario hace clic en el botón **Cancelar** en esta caja de diálogo, la ejecución se detiene y la variable sistema OK toma el valor 0.*

***Nota**: este comando no puede utilizarse en el marco de una conexión con el motor SQL interno de 4D.*

Si no pasa el parámetro proyecto, 4D muestra la caja de diálogo de exportación ODBC, permitiendo al usuario configurar la operación. Para mayor información sobre esta caja de diálogo, consulte el manual de Diseño.
*Si pasa en el parámetro proyecto un BLOB que contiene un proyecto de exportación ODBC válido, la exportación se efectuará directamente, sin intervención del usuario. Para hacer esto, simplemente necesita cargar un proyecto guardado previamente en el disco en el campo o variable BLOB que pasa en el parámetro proyecto, utilizando el comando **DOCUMENT TO BLOB**. Los proyectos de exportación ODBC se guardan desde la caja de diálogo de exportación ODBC.*
*También puede utilizar el comando **EXPORT ODBC** con un parámetro proyecto vacío y el parámetro opcional *, luego almacenar el parámetro proyecto en un campo BLOB (ver a continuación). Esta solución permite, por una parte, almacenar el proyecto con el archivo de datos y por otra parte, evitar la fase de carga del disco al BLOB.*

Los parámetros de conexión (nombre de la fuente, usuario y contraseña) se incluyen en el BLOB proyecto.

***Nota:** consulte el comando **EXPORT DATA** para ver un ejemplo relacionado con la definición de un proyecto vacío. Note que los proyectos generados en la caja de diálogo de importación ODBC no son compatibles con los comandos o la caja de diálogo de importación estándar de 4D.*
*El parámetro opcional *, si se especifica, muestra la caja de diálogo de exportación de datos ODBC con los parámetros eventualmente definidos en proyecto. Este funcionamiento permite utilizar un proyecto predefinido mientras tiene la posibilidad de modificar uno o varios parámetros. Además, en este caso el parámetro proyecto contiene los parámetros del "nuevo" proyecto después del cierre de la caja de diálogo. Entonces puede almacenarlo en un campo BLOB, en un archivo en disco, etc.*

Variables y conjuntos del sistema

*Si el usuario hace clic en **Cancelar** en una de las dos cajas de diálogo (para seleccionar la fuente de datos o los parámetros de exportación), la variable sistema OK toma el valor 0. Si la exportación se lleva a cabo correctamente, la variable sistema OK toma el valor 1.*

EXPORT SYLK

EXPORT SYLK ({tabla ;} doc)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual exportar datos, o Tabla por defecto, si se omite
doc	Cadena	➡	Documento SYLK para recibir los datos

Descripción

*El comando **EXPORT SYLK** escribe los datos de los registros de la selección actual de tabla en el proceso actual. Los datos se escriben en documento, un documento Sylk Windows o Macintosh, en el disco.*

La operación de exportación se efectúa por el formulario de salida actual. La operación de exportación escribe los campos y las variables basado en el orden de entrada del formulario de salida. Por esta razón, utilice un formulario de salida que sólo contenga los campos u objetos editables que quiera exportar. No ubique botones u otros objetos en el formulario de exportación. Los objetos de subformulario se ignoran.

Un evento On Load se envía al método del formulario por cada registro exportado. Utilice este evento para definir las variables que podría utilizar en el formulario de exportación.

El parámetro documento puede dar nombre a un documento nuevo o existente. Si documento tiene el mismo nombre que un documento existente, el documento existente se sobrescribe. El documento puede incluir una ruta de acceso que contenga los nombres de los volúmenes y las carpetas. Si pasa una cadena vacía, aparece la caja de diálogo estándar de guardar archivos. Si el usuario cancela este diálogo, se cancela la operación de exportación, y la variable sistema OK toma el valor 0.

*Un termómetro de progreso aparece durante la exportación. El usuario puede cancelar la operación haciendo clic en el botón Detener. Si la exportación termina con éxito, la variable sistema OK toma el valor 1. En caso de error o si la operación se interrumpe, la variable OK toma el valor 0. El termómetro puede ocultarse con el comando **MESSAGES OFF**.*

*El comando utiliza por defecto el conjunto de caracteres UTF-8. Los documentos en formato SYLK utilizan por lo general el conjunto de caracteres ISO-8859-1, puede utilizar el comando **USE CHARACTER SET** para especificar el conjunto de caracteres apropiado.*

*Durante la utilización de **EXPORT SYLK**, el delimitador de campos por defecto es el carácter de tabulación (código 9). El delimitador del registro por defecto es el retorno de carro (código 13) en OS X y el retorno de carro+retorno a la línea (código 13 + código 10) bajo Windows. Puede modificar estos valores asignando nuevos valores a las dos **Variables sistema** FldDelimit y RecDelimit. El usuario puede cambiar estos valores en la caja de diálogo de exportación del entorno Diseño. Tenga en cuenta que si los campos exportados contienen caracteres definidos como delimitadores de campos o de registros, estos caracteres se reemplazan automáticamente con espacios en el archivo exportado, con el fin de no perturbar el proceso de importación.*

Ejemplo

El siguiente ejemplo exporta datos a un documento SYLK. El método primero define el formulario de salida de manera que los datos se exporten por el formulario correcto, luego efectúa la exportación:

```
FORM SET OUTPUT([Personas];"Exportar")
EXPORT SYLK([Personas];"Nuevas _Personas.slk")  ` Exporta al documento "Nuevas_Personas.slk"
```

Variables y conjuntos del sistema

OK toma el valor 1 si la exportación termina con éxito; de lo contrario, toma el valor 0.

EXPORT TEXT

EXPORT TEXT ({tabla ;} doc)

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla desde la cual exportar datos o Tabla por defecto, si se omite
doc	Cadena	⇒	Documento texto para recibir los datos

Descripción

*El comando **EXPORT TEXT** escribe datos de los registros de la selección actual de la tabla en el proceso actual. Los datos se escriben en documento, un documento de texto Windows o Macintosh en el disco.*

La operación de exportación se efectúa por el formulario de salida actual. La operación de exportación escribe los campos y las variables basado en el orden de entrada del formulario de salida. Por esta razón, utilice un formulario de salida que sólo contenga los campos u objetos editables que quiera exportar. No ubique botones u otros objetos en el formulario de exportación. Los objetos de subformulario se ignoran.

Un evento On Load se envía al método del formulario por cada registro exportado. Utilice este evento para definir las variables que podría utilizar en el formulario de exportación.

El parámetro documento puede dar nombre a un documento nuevo o existente. Si documento tiene el mismo nombre que un documento existente, el documento existente se sobrescribe. El documento puede incluir una ruta de acceso que contenga los nombres de los volúmenes y las carpetas. Si pasa una cadena vacía, aparece la caja de diálogo estándar de guardar archivos. Si el usuario cancela este diálogo, se cancela la operación de exportación, y la variable sistema OK toma el valor 0.

*Un termómetro de progreso aparece durante la exportación. El usuario puede cancelar la operación haciendo clic en el botón Detener. Si la exportación termina con éxito, la variable sistema OK toma el valor 1. En caso de error o si la operación se interrumpe, la variable OK toma el valor 0. El termómetro puede ocultarse con el comando **MESSAGES OFF**.*

*El comando utiliza por defecto el conjunto de caracteres UTF-8. Puede utilizar el comando **USE CHARACTER SET** para modificar este conjunto de caracteres.*

*Durante la utilización de **EXPORT TEXT**, el delimitador de campos por defecto es el carácter de tabulación (code 9). El delimitador de registros por defecto es el retorno de carro (code 13) bajo OS X y el retorno de carro+retorno a la línea (código 13+código 10) bajo Windows. Puede cambiar estos valores por defecto asignando nuevos valores a las **Variables sistema**: **FldDelimit** y **RecDelimit**. El usuario puede cambiar los valores por defecto en la caja de diálogo de exportación del entorno Aplicación. Tenga en cuenta que si los campos exportados contienen caracteres definidos como delimitadores de campo o de registro, estos caracteres se reemplazan automáticamente con espacios en el archivo exportado, con el fin de no perturbar el proceso de importación.*

Ejemplo

Este ejemplo exporta datos a un documento texto. El método primero comienza por definir el formulario de salida se manera que los datos sean exportados por el formulario correcto, cambia los delimitadores, luego efectúa la exportación:

```
FORM SET OUTPUT([Personas];"Exportar")
FldDelimit:=27 ` Definir carácter delimitador de campos: Escape
RecDelimit:=10 ` Definir carácter delimitador de registros: Line Feed
EXPORT TEXT([Personas];"Nuevas_Personas.txt") ` Exportación del documento "Nuevas_Personas.txt"
```

Variables y conjuntos del sistema

OK toma el valor 1 si la exportación termina con éxito; de lo contrario, toma el valor 0.

IMPORT DATA

IMPORT DATA (nomArchivo {; proyecto {; *}})

Parámetro	Tipo		Descripción
nomArchivo	Cadena	➡	Ruta de acceso y nombre del archivo a importar
proyecto	Variable texto, BLOB variable	➡	Contenido del proyecto de importación
		⬅	Nuevo contenido del proyecto de importación (si se pasa el parámetro *)
*	Operador	➡	Visualización de la caja de diálogo de importación y actualización del proyecto

Descripción

*El comando **IMPORT DATA** importa los datos ubicados en el archivo nomArchivo. 4D puede importar los datos en los siguientes formatos: Texto, Texto de longitud fija, XML, SYLK, DIF, DBF (dBase) y 4D.*

*Si pasa una cadena vacía en nomArchivo, **IMPORT DATA** muestra la caja de diálogo estándar de apertura de archivos, permitiendo al usuario definir el nombre, tipo, y ubicación del archivo a importar. Una vez aceptada la caja de diálogo, la variable sistema **Documento** contiene la ruta de acceso y el nombre del archivo. Si el usuario hace clic en **Cancelar**, se detiene la ejecución del comando y la variable sistema **OK** toma el valor 0.*

*El parámetro opcional proyecto le permite utilizar un proyecto para importar datos. Cuando pasa este parámetro, la importación se realiza directamente, sin intervención del usuario (a menos que utilice la opción *, ver a continuación). Si no pasa este parámetro, aparece la caja de diálogo de importación. El usuario puede definir sus parámetros de importación o cargar un proyecto de importación existente.*

Un proyecto de importación contiene todos los parámetros de importación, tales como las tablas y campos en las cuales importar, los delimitadores a utilizar, etc. En el parámetro proyecto, puede pasar una variable Texto con XML o una variable Texto con una referencia a un elemento DOM pre existente o un BLOB. Los proyectos pueden crearse por programación (proyectos con formato XML únicamente) o cargando parámetros definidos previamente en la caja de diálogo de importación. En el último caso, tiene dos soluciones disponibles:

- *Utilice el comando **IMPORT DATA** con un parámetro proyecto vacío y el parámetro opcional *, luego guarde el parámetro proyecto resultante en un campo Texto o BLOB (ver a continuación). Esta solución le permite guardar el proyecto con el archivo de datos.*
- *Guarde el proyecto en el disco, luego cárguelo, utilizando por ejemplo el comando **DOM Parse XML source**, y pase su referencia en el parámetro proyecto.*

***Nota de compatibilidad**: a partir de la versión 12 de 4D, los proyectos de importación son codificados en XML. 4D puede abrir los proyectos de importación generados con las versiones anteriores de 4D (formato BLOB), sin embargo los proyectos creados a partir de la v12 no pueden abrirse con una versión 11 o anterior. Recomendamos utilizar variables Texto para manipular los archivos de importación.*

*El parámetro opcional *, si se especifica, hace que aparezca la caja de diálogo de importación con los parámetros definidos en proyecto. Esta característica le permite utilizar un proyecto predefinido, mientras tiene aún la posibilidad de modificar uno o más parámetros. Además, el parámetro proyecto contiene, después de cerrar la caja de diálogo de importación, los parámetros del "nuevo" proyecto. Entonces puede almacenar el nuevo proyecto en un campo BLOB, en disco, etc.*

Si la importación fue exitosa, la variable sistema OK toma el valor 1.

***Nota:** consulte el comando **EXPORT DATA** para ver un ejemplo sobre la definición de un proyecto vacío.*

Variables y conjuntos del sistema

*Si el usuario hace clic en **Cancelar** en la caja de diálogo estándar de guardar archivos o de importación, la variable sistema OK toma el valor 0. Si la importación fue exitosa, la variable sistema OK toma el valor 1.*

IMPORT DIF

IMPORT DIF ({tabla ;} doc)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla en la cual importar datos, o Tabla por defecto, si se omite
doc	Cadena	➡	Documento DIF a importar

Descripción

*El comando **IMPORT DIF** lee datos de documento, un documento DIF Windows o Macintosh y los escribe en tabla creando nuevos registros para esa tabla.*

La operación de importación se realiza por intermedio del formulario de entrada actual. La operación de importación lee los campos y las variables en función del plano de los objetos en el formulario de entrada. Por esta razón, debe ser muy cuidadoso en cuanto al orden de los objetos texto (campos y variables) en el formulario. El primer objeto en el cual los datos serán importados debe estar al fondo del formulario, etc. Si el número de campos o variables en el formulario no corresponde al número de campos a importar, los campos extras son ignorados. Un formulario de entrada utilizado para importar no puede contener botones. Los objetos de subformulario son ignorados.

***Nota:** una manera de asegurar que los datos se importen en los objetos correctos es seleccionar el objeto en el cual debe importarse el primer campo y moverlo al primer plano. Continúe moviendo los campos y variables al primer plano, en orden, asegurándose de que tenga un campo o variable para cada campo a importar.*

Un evento On Validate se envía al método de formulario para cada registro que se importa. Utilice este evento para copiar los datos de las variables a los campos, si utiliza las variables en el formulario de importación.

El parámetro documento puede incluir una ruta de acceso que contenga los nombres de volúmenes y carpetas. Si pasa una cadena vacía, aparece la caja de diálogo estándar de apertura de archivos. Si el usuario cancela este diálogo, se cancela la operación de importación, y la variable sistema OK toma el valor 0.

*Un termómetro de progreso aparece durante la importación. El usuario puede cancelar la operación haciendo clic en el botón Detener. Los registros que ya hayan sido importados no serán removidos si el usuario presiona el botón Detener. Si la importación termina con éxito, la variable sistema OK toma el valor 1. Si ocurre un error o se interrumpe la operación, la variable sistema toma el valor 0. El termómetro puede ocultarse con el comando **MESSAGES OFF**.*

*El comando utiliza por defecto el conjunto de caracteres UTF-8. Los documentos con formato DIF utilizan por lo general el conjunto de caracteres IBM437, puede ser necesario utilizar el comando **USE CHARACTER SET** para definir el conjunto de caracteres apropiado.*

*Durante la utilización de **IMPORT DIF**, el delimitador de campos por defecto es el carácter de tabulación (código 9). El delimitador del registro por defecto es el retorno de carro (código 13). Puede modificar estos valores asignando nuevos valores a las dos variables sistema FldDelimit y RecDelimit. El usuario puede cambiar estos valores en la caja de diálogo de exportación del entorno Diseño. Como los campos Texto pueden contener retornos de carro, sea cuidadoso si utiliza el retorno de carro como delimitador entre los campos a exportar.*

Ejemplo

El siguiente ejemplo importa datos de un documento DIF. El método comienza definiendo el formulario de entrada de manera que los datos se importen por el formulario correcto, luego realiza la importación:

```
FORM SET INPUT([Personas];"Importar")
IMPORT DIF([Personas];"Nuevas_Personas.dif") ` Importación del documento "Nuevas_Personas.dif"
```

Variables y conjuntos del sistema

OK toma el valor 1 si la importación termina con éxito; de lo contrario, toma el valor 0.

IMPORT ODBC

IMPORT ODBC (tablaFuente {; proyecto {; *}})

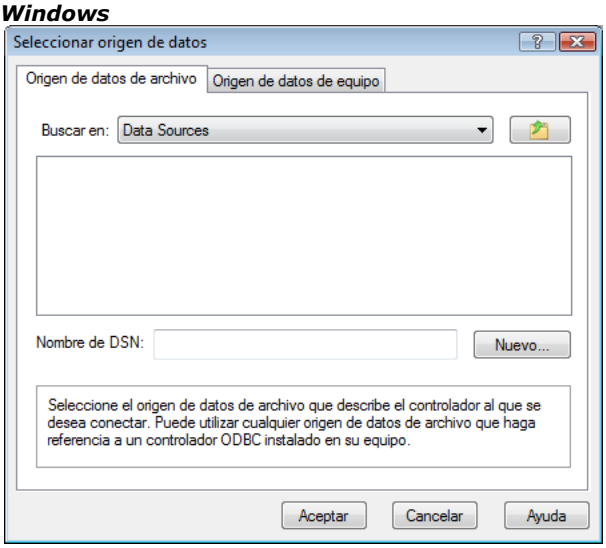
Parámetro	Tipo		Descripción
tablaFuente	Cadena	➡	Nombre de la tabla en la fuente de datos ODBC
proyecto	BLOB	➡	Contenido del proyecto de importación
		⬅	Nuevo contenido del proyecto de importación (si se pasa *)
*	Operador	➡	Visualización de la caja de diálogo de importación y actualización del proyecto

Descripción

*El comando **IMPORT ODBC** permite importar datos desde la tabla tablaFuente de una fuente ODBC externa.*

*Si llama al comando **IMPORT ODBC** fuera de una conexión previamente abierta con la ayuda del comando , se muestra la caja de diálogo de selección de la fuente de datos ODBC:*

Windows



Mac OS

*Si el usuario hace clic en el botón **Cancelar** en esta caja de diálogo, se detiene la ejecución y la variable sistema OK toma el valor 0.*

***Nota:** este comando no puede utilizarse en el caso de conexiones con el núcleo interno SQL de 4D.*

Si no pasa el parámetro proyecto, 4D muestra la caja de diálogo de importación ODBC, permitiendo al usuario configurar la operación. Para mayor información sobre esta caja de diálogo, consulte el manual de Diseño.

*Si pasa en el parámetro proyecto un BLOB que contiene un proyecto de importación ODBC válido, la importación se efectúa directamente, sin intervención del usuario. Para hacer esto, simplemente necesita cargar un proyecto que haya sido guardado previamente en el disco en el campo o variable BLOB que pasa en el parámetro proyecto, utilizando el comando **DOCUMENT TO BLOB**. Los proyectos de importación ODBC se guardan desde la caja de diálogo de importación ODBC.*
*También puede utilizar el comando **IMPORT ODBC** con un parámetro proyecto vacío y el parámetro opcional *, luego almacenar el parámetro proyecto en un campo BLOB (ver a continuación). Esta solución permite, por una parte, almacenar el proyecto con el archivo de datos y por otra parte, evitar la fase de carga del disco al BLOB.*

***Nota:** consulte el comando **EXPORT DATA** para ver un ejemplo de definición de un proyecto vacío. Note que los proyectos generados en la caja de diálogo de importación ODBC no son compatibles con los comandos o la caja de diálogo de importación estándar de 4D.*

*El parámetro opcional *, si se especifica, muestra la caja de diálogo de importación de datos ODBC con los parámetros eventualmente definidos en proyecto. Este funcionamiento permite utilizar un proyecto predefinido mientras tiene la posibilidad de modificar uno o más parámetros. Además, en este caso el parámetro proyecto contiene los parámetros del "nuevo" proyecto después del cierre de la caja de diálogo. Entonces usted puede almacenarlo en un campo BLOB, en un archivo en disco, etc.*

Variables y conjuntos del sistema

*Si el usuario hace clic en **Cancelar** en una de las dos cajas de diálogo (para seleccionar la fuente de datos o los parámetros de importación), la variable sistema OK toma el valor 0. Si la importación se lleva a cabo correctamente, la variable sistema OK toma el valor 1.*

IMPORT SYLK

IMPORT SYLK ({tabla ;} doc)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla en la cual importar los datos, o Tabla por defecto, si se omite
doc	Cadena	➡	Documento SYLK a importar

Descripción

*El comando **IMPORT SYLK** lee los datos de documento, un documento SYLK Windows o Macintosh, y los escribe en tabla creando nuevos registros.*

La operación de importación se realiza por intermedio del formulario de entrada actual. La operación de importación lee los campos y las variables en función del plano de los objetos en el formulario de entrada. Por esta razón, debe ser muy cuidadoso en cuanto al orden de los objetos texto (campos y variables) en el formulario. El primer objeto en el cual los datos serán importados debe estar al fondo del formulario, etc. Si el número de campos o variables en el formulario no corresponde al número de campos a importar, los campos extras son ignorados. Un formulario de entrada utilizado para importar no puede contener botones. Los objetos de subformulario son ignorados.

***Nota:** una forma de asegurar que los datos se importen en los objetos correctos es seleccionar el objeto en el cual se debe importar el primer campo y moverlo al primer plano. Continúe moviendo los campos y variables al primer plano, en orden, asegurándose de que tenga un campo o variable para cada campo a importar.*

Un evento On Validate se envía al método de formulario para cada registro que se importa. Utilice este evento para copiar los datos de las variables a los campos, si utiliza las variables en el formulario de exportación.

El parámetro documento puede incluir una ruta de acceso a los nombres de volúmenes y carpetas. Si pasa una cadena vacía, aparece la caja de diálogo estándar de apertura de archivos. Si el usuario cancela este diálogo, se cancela la operación de importación, y la variable sistema OK toma el valor 0.

*Un termómetro de progreso aparece durante la importación. El usuario puede cancelar la operación haciendo clic en el botón Detener. Los registros que ya hayan sido importados no serán removidos si el usuario presiona el botón Detener. Si la importación termina con éxito, la variable sistema OK toma el valor 1. Si ocurre un error o se interrumpe la operación, la variable sistema toma el valor 0. El termómetro puede ocultarse con el comando **MESSAGES OFF**.*

*El comando utiliza por defecto el conjunto de caracteres UTF-8. Los documento en formato SYLK utilizan por lo general el conjunto de caracteres ISO-8859-1, puede ser necesario utilizar el comando **USE CHARACTER SET** para definir el conjunto de caracteres apropiado.*

*Durante la utilización de **IMPORT SYLK**, el delimitador de campos por defecto es el carácter de tabulación (código 9). El delimitador del registro por defecto es el retorno de carro (código 13). Puede modificar estos valores asignando nuevos valores a las dos **Variables sistema** FldDelimit y RecDelimit. El usuario puede cambiar estos valores en la caja de diálogo de exportación del entorno Diseño. Como los campos Texto pueden contener retornos de carro, sea cuidadoso si utiliza el retorno de carro como delimitador entre los campos a exportar.*

Ejemplo

El siguiente ejemplo importa datos de un documento SYLK. El método comienza por definir el formulario de entrada de manera que los datos sean importados por el formulario correcto, luego realiza la importación:

```
FORM SET INPUT([Personas];"Importar")
IMPORT SYLK([Personas];"Nuevas_Personas.slk")  ` Importar del documento "Nuevas_Personas.slk"
```

Variables y conjuntos del sistema

OK toma el valor 1 si la importación termina con éxito; de lo contrario, toma el valor 0.

IMPORT TEXT

IMPORT TEXT ({tabla ;} doc)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla en la cual importar datos, o Tabla por defecto, si se omite
doc	Cadena	➡	Documento texto a importar datos

Descripción

*El comando **IMPORT TEXT** lee los datos de documento, un documento texto Windows o Macintosh, y los escribe en tabla creando nuevos registros para esa tabla.*

La operación de importación se efectúa a través del formulario de entrada actual. La operación de importación lee los campos y variables en función de las capas de objetos en el formulario de entrada. Por esta razón, debe ser muy cuidadoso en cuanto al orden de los objetos (campos y variables) en el formulario. El primer objeto en el cual los datos serán importados debe estar al fondo del formulario, etc. Si el número de campos o variables en el formulario no corresponde al número de campos a importar, los campos extras son ignorados. Un formulario de entrada utilizado para importar no puede contener botones. Los objetos de subformulario son ignorados.

***Nota:** una manera de asegurarse de que los datos sean importados en el objeto correcto es seleccionar el objeto en el cual será importado el primer campo y moverlo al primer plano. Continúe moviendo los campos y variables al primer plano en orden, asegurándose de que tenga un campo o variable para cada campo a importar.*

El evento On Validate es enviado al método de formulario para cada registro importado. Utilice este evento para copiar los datos de las variables en los campos, si utiliza variables en el formulario de importación.

El parámetro documento puede incluir una ruta de acceso a los nombres de los volúmenes y carpetas. Si pasa una cadena vacía, aparece la caja de diálogo estándar de apertura de archivos. Si el usuario cancela este diálogo, la operación de importación se cancela, y la variable sistema OK toma el valor 0.

*Un termómetro de progreso aparece durante la importación. El usuario puede cancelar la operación haciendo clic en el botón Detener. Los registros que ya han sido importados no se eliminarán si el usuario presiona el botón Detener. Si la importación se completa con éxito, la variable sistema OK toma el valor 1. En caso de error o si la operación se interrumpe, la variable OK toma el valor 0. El termómetro puede ocultarse con el comando **MESSAGES OFF**.*

*El comando utiliza por defecto el conjunto de caracteres UTF-8. Usted puede utilizar el comando **USE CHARACTER SET** para modificar este conjunto de caracteres.*

*Durante la utilización de **IMPORT TEXT**, el delimitador de campos por defecto es el carácter de tabulación (code 9). El delimitador de registros por defecto es el retorno de carro (code 13). Puede cambiar estos valores por defecto asignando nuevos valores a las **Variables sistema**: FldDelimit y RecDelimit. El usuario puede cambiar los valores por defecto en la caja de diálogo de importación del entorno Diseño. Los campos texto pueden contener retornos de carro, por lo tanto, tenga cuidado cuando utilice como delimitador un retorno de carro si está importando campos texto.*

Ejemplo

El siguiente ejemplo importa datos de un documento texto. El método primero define el formulario de entrada de manera que los datos sean importados al formulario correcto, cambia el delimitador de variables 4D, luego efectúa la importación:

```
FORM SET INPUT([Personas];"Importar")
FldDelimit:=27 ` Definir carácter delimitador de campos: Escape
RecDelimit:=10 ` Definir carácter delimitador de registros: Line Feed
IMPORT TEXT([Personas];"Nuevas_Personas.txt") ` Importación del documento “Nuevas_Personas.txt”
```

Variables y conjuntos del sistema

OK toma el valor 1 si la importación termina con éxito; de lo contrario, toma el valor 0.

Impresión

- *Integración del driver PDFCreator bajo Windows*
- *ACCUMULATE*
- *BLOB to print settings*
- *BREAK LEVEL*
- *CLOSE PRINTING JOB*
- *Get current printer*
- *Get print marker*
- *GET PRINT OPTION*
- *Get print preview*
- *GET PRINTABLE AREA*
- *GET PRINTABLE MARGIN*
- *Get printed height*
- *Is in print preview*
- *Level*
- *OPEN PRINTING JOB*
- *PAGE BREAK*
- *PAGE SETUP*
- *Print form*
- *PRINT LABEL*
- *Print object*
- *PRINT OPTION VALUES*
- *PRINT RECORD*
- *PRINT SELECTION*
- *PRINT SETTINGS*
- *Print settings to BLOB*
- *PRINTERS LIST*
- *Printing page*
- *SET CURRENT PRINTER*
- *SET PRINT MARKER*
- *SET PRINT OPTION*
- *SET PRINT PREVIEW*
- *SET PRINTABLE MARGIN*
- *Subtotal*

Integración del driver PDFCreator bajo Windows

El soporte para la impresión PDF difiere dependiendo de la versión de Windows:

- *para Windows 8 y versiones anteriores, debe utilizar el driver PDFCreator.*
- *a partir de Windows 10, está integrado.un driver nativo Microsoft*

***Nota:** en Mac OS, la impresión PDF es soportada nativamente por el sistema.*

Windows 8 y versiones anteriores

El soporte de las impresiones PDF en Windows se basa en el driver PDFCreator para ofrecer funciones de impresión PDF simples y funcionales. Los comandos ***GET PRINT OPTION*** *y* ***SET PRINT OPTION*** *hacen uso de este driver.*

PDFCreator es un driver gratuito (OpenSource) que se rije por la licencia AFPL (Aladdin Free Public License). Para utilizar el driver PDFCreator, debe descargar e instalar la versión apropiada en su entorno, no es instalada por defecto por 4D. Debe tener derechos de acceso de administrador para poder instalar el driver. Puede descargar el PDFCreator aquí:
http://sourceforge.net/projects/pdfcreator/files/PDFCreator/

***Atención:** debe utilizar una versión de PDFCreator que sea **compatible con 4D**. Para conocer las versiones compatibles y certificadas de PDFCreator, por favor consulte las matrices de certificación de los productos 4D, disponibles en la **Página Resources** (**sección Compatibilidad**) del sitio web de 4D Web.*

Durante la instalación, una nueva impresora virtual llamada por defecto "PDFCreator" se instala en su sistema. Puede cambiar este nombre si lo prefiere.

A partir de Windows 10

Windows 10 incluye un driver nativo PDF que permite a 4D crear directamente los PDFs, sin necesidad de utilizar un controlador de terceros como PDFCreator.

El nombre del driver es "Microsoft Print to PDF".
Este es un ejemplo de cómo crear un documento PDF en Windows 10 utilizando los comandos de impresión 4D:

```
$pdfpath:=System folder(Desktop)+"test.pdf"

$pdfprintername:="Microsoft Print to PDF"
ARRAY TEXT($name1;0)
PRINTERS LIST($name1)
If(Find in array($name1;$pdfprintername)>0)
    SET CURRENT PRINTER($pdfprintername)
    SET PRINT OPTION(Destination option;2;$pdfpath)
    ALL RECORDS([Table_1])
    PRINT SELECTION([Table_1];*)
    SET CURRENT PRINTER("")
End if
```

ACCUMULATE

ACCUMULATE (objeto {; objeto2 ; ... ; objetoN})

Parámetro	Tipo		Descripción
objeto	Campo, Variable	➡	Campo o variable de tipo numérico a acumular

Descripción

ACCUMULATE especifica los campos o variables a acumular en un informe realizado utilizando **PRINT SELECTION**.

Debe ejecutar **BREAK LEVEL** y **ACCUMULATE** antes de cada informe para el cual quiera utilizar rupturas. Estos comandos activan el proceso de rupturas para un informe. Ver la explicación del comando **Subtotal**.

Utilice **ACCUMULATE** cuando quiera incluir subtotales para tal los campos o variables numéricas en un informe. **ACCUMULATE** le indica a 4D que almacene los subtotales para cada elemento especificado en objeto. Los subtotales se acumulan para cada nivel de ruptura especificado por el comando **BREAK LEVEL**.

Ejecute **ACCUMULATE** antes de imprimir un informe con **PRINT SELECTION**.

Utilice la función **Subtotal** en el método de formulario o en un método de objeto para devolver el subtotal de uno de los objetos especificados en objeto.

Ejemplo

Ver el ejemplo del comando **BREAK LEVEL**.

BLOB to print settings

BLOB to print settings (confImpr {; params}) -> Resultado

Parámetro	Tipo		Descripción
confImpr	BLOB	➡	BLOB que contiene la configuración de impresión
params	Entero largo	➡	0=Restaura valores guardados para el número de copias y rango de páginas, 1=Restablece los valores predeterminados
Resultado	Entero largo	⬅	Código de estado: 1=operación exitosa, 0=no hay impresora actual, -1=parámetros incorrectos, 2=impresora modificada

Descripción

*El comando **BLOB to print settings** reemplaza los parámetros de impresión actuales de 4D por los parámetros almacenados en el BLOB confImpr. Este BLOB debe haber sido generado por el comando **Print settings to BLOB** o por el comando 4D Pack 4D Pack (ver abajo).*

El parámetro params le permite definir cómo manejar los parámetros básicos para el "número de copias" y el "intervalo de impresión":

- *Si pasa 0 u omite este parámetro, los valores almacenados en el BLOB se utilizan para la impresión.*
- *Si pasa 1, los valores se restablecen a los valores predeterminados: el número de copias se establece en 1 y el intervalo de páginas se establece en "todas las páginas".*

*Los parámetros de impresión se aplican a la impresora actual y durante toda la sesión, siempre y cuando ningún comando como **PAGE SETUP**, [#cmd id="733"/] o **PRINT SELECTION** sin el parámetro > los modifique. Los parámetros definidos se utiliza particularmente para los comandos **PRINT SELECTION**, **PRINT LABEL**, **PRINT RECORD**, [#cmd id="5"/] y **QR REPORT**, así como también para los comandos de impresión en los menú de 4D, incluyendo los del entorno Diseño.*

*Los comandos **PRINT SELECTION**, **PRINT LABEL** y **PRINT RECORD** deben ser llamados con el parámetro > (si aplica) para que los parámetros definidos por **BLOB to print settings** se mantengan.*

El comando devuelve uno de los siguientes códigos de estado:

- *-1: el BLOB es incorrecto,*
- *0: ninguna impresora actual está seleccionada (en este caso, el comando no hace nada),*
- *1: el BLOB se ha cargado correctamente,*
- *2: el BLOB se ha cargado correctamente, pero el nombre de la impresora actual ha cambiado(*)*

  ***Nota:** Code (2) siempre se devuelve si el BLOB fue creado por el comando 4D Pack , incluso si el nombre de la impresora no cambió, ya que esta información no se incluyó en los BLOBs 4D Pack.*

(*) Los parámetros dependen de la impresora actual seleccionada en el momento en que el BLOB se guardó. La aplicación de estos valores en otra impresora es soportada si ambas impresoras son del mismo modelo. Si las impresoras son diferentes, se restaurarán sólo los parámetros comunes.

Windows / OS X

El BLOB confImp se puede guardar y leer en ambas plataformas. Sin embargo, incluso si ciertos parámetros de impresión son comunes, algunos otros son específicos de la plataforma y dependen de los controladores de impresión y de las versiones del sistema operativo. Si el mismo BLOB confImp se comparte entre ambas plataformas, es posible que pierda partes de información.

Cuando se utiliza en un entorno heterogéneo, con el fin de restaurar el máximo de parámetros de impresión disponibles para cada plataforma (y no sólo la parte común), se recomienda que maneje dos BLOBs confImp, uno para cada plataforma.

Compatibilidad con los comandos 4D Pack

*Los BLOBs de parámetros de impresión generados con el comando 4D Pack [#cmd id="61955"/] pueden ser cargados y utilizados por el comando **BLOB to print settings**. Tenga en cuenta sin embargo, que si son guardados con [#cmd id="1433"/], se convierten y no se abrirán más con . El comando **BLOB to print settings** permite almacenar más información que el comando .*

Ejemplo

Usted desea aplicar la configuración de impresión guardada en el disco para el contexto de impresión 4D actual:

```
C_BLOB(curSettings)
DOCUMENT TO BLOB(Get 4D folder(Active 4D Folder)+"current4Dsettings.blob";curSettings)
 //current4Dsettings ha sido creado por Print settings to BLOB
$err:=BLOB to print settings(curSettings;0)
Case of
   :($err=1)
 //todo está OK
   :($err=2)
      CONFIRM("¡La impresora ha cambiado!\n\n¿Revisar los parámetros de impresión?")
      If(OK=1)
         PRINT SETTINGS
      End if
```

```
    :($err=0)
        ALERT("No hay impresora actual.")
    :($err=-1)
        ALERT("Archivo de configuración no valido.")
End case
```

BREAK LEVEL

BREAK LEVEL (nivel {; saltoPag})

Parámetro	Tipo		Descripción
nivel	Entero largo	➡	Número de niveles de ruptura
saltoPag	Entero largo	➡	Nivel del salto de página

Descripción

BREAK LEVEL especifica el número de niveles de ruptura en un informe realizado utilizando **PRINT SELECTION**.

Debe ejecutar **BREAK LEVEL** y **ACCUMULATE** antes de cada informe en el cual quiera utilizar rupturas. Estos comandos activan el proceso de rupturas para un informe. Ver la explicación para el comando **Subtotal**.

El parámetro nivel indica el último nivel de ruptura para el que quiere realizar procesos de rupturas. Debe haber ordenado los registros con al menos ese número de niveles. Si ha ordenado más niveles, estos niveles serán impresos como están ordenados, pero no serán procesados para rupturas.

Cada nivel de ruptura generado imprimirá las áreas de rupturas y de encabezado correspondientes en el formulario. Debe haber en el formulario al menos tantas áreas de ruptura como el número que pasó en nivel. Si hay más áreas de ruptura, serán ignoradas y no se imprimirán.

El segundo parámetro, opcional, saltoPag, se utiliza para provocar saltos de página durante la impresión.

Ejemplo

El siguiente ejemplo imprime un informe con dos niveles de ruptura. La selección es ordenada en cuatro niveles, pero el comando **BREAK LEVEL** especifica sólo dos niveles de ruptura. Un campo se acumula con el comando **ACCUMULATE**:

```
ORDER BY([Emp]Dept;>;[Emp]Title;>;[Emp]Apellido;>;[Emp]Nombre;>) ` Ordenar en cuatro niveles
BREAK LEVEL(2) ` Fijar dos niveles de ruptura(Dept y Title)
ACCUMULATE([Emp]Salario) ` Acumular los salarios
FORM SET OUTPUT([Emp];"Dept salario") ` Seleccionar el formulario a imprimir
PRINT SELECTION([Emp]) ` Imprimir el informe
```

CLOSE PRINTING JOB

CLOSE PRINTING JOB

Este comando no requiere parámetros

Descripción

*El comando **CLOSE PRINTING JOB** permite cerrar el trabajo de impresión previamente abierto por el comando **OPEN PRINTING JOB** y enviar a la impresora actual el documento de impresión eventualmente construido.*

Una vez ejecutado este comando, la impresora nuevamente está disponible para otros trabajos de impresión.

Get current printer

Get current printer -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Cadena*		*Nombre de la impresora actual*

Descripción

El comando ***Get current printer****devuelve el nombre de la impresora actual definida en la aplicación 4D. Por defecto, al inicio de 4D, la impresora actual es la impresora definida en el sistema.*

Si la impresora actual es administrada utilizando un servidor de impresión (spooler), se devuelve la ruta de acceso completa (bajo Windows) o el nombre del spooler (bajo Mac OS).

Para obtener la lista de impresoras disponibles como también información adicional, utilice el comando ***PRINTERS LIST****. Para modificar la impresora actual, utilice el comando* ***SET CURRENT PRINTER****.*

Nota: *cuando la constante Generic PDF driver se utiliza con* ***SET CURRENT PRINTER****,* ***Get current printer*** *devuelve "_4d_pdf_printer" o el nombre actual del driver PDF, si aplica (opción no disponible con 4D 32 bits).*

Variables y conjuntos del sistema

Si no está instalada ninguna impresora, la variable sistema OK toma el valor 0. De lo contrario, toma el valor 1.

Get print marker

Get print marker (markNum) -> Resultado

Parámetro	Tipo		Descripción
markNum	Entero largo	➡	Número de marcador
Resultado	Entero largo	⮌	Posición del marcador

Descripción

*El comando **Get print marker** permite obtener la posición actual de un marcador durante una impresión.*
Este comando puede utilizarse en dos contextos:

- *Durante el evento de formulario **On Header**, en el contexto de los comandos **PRINT SELECTION** y **PRINT RECORD**.*
- *Durante el evento de formulario **On Printing Detail**, en el contexto del comando **Print form**.*

Se devuelven las coordenadas en píxeles (1 píxel = 1/72 pulgadas).
Pase una de las constantes del tema Área de formulario en el parámetro nummarc:

Constante	Tipo	Valor
Form break0	Entero largo	300
Form break1	Entero largo	301
Form break2	Entero largo	302
Form break3	Entero largo	303
Form break4	Entero largo	304
Form break5	Entero largo	305
Form break6	Entero largo	306
Form break7	Entero largo	307
Form break8	Entero largo	308
Form break9	Entero largo	309
Form detail	Entero largo	0
Form footer	Entero largo	100
Form header	Entero largo	200
Form header1	Entero largo	201
Form header10	Entero largo	210
Form header2	Entero largo	202
Form header3	Entero largo	203
Form header4	Entero largo	204
Form header5	Entero largo	205
Form header6	Entero largo	206
Form header7	Entero largo	207
Form header8	Entero largo	208
Form header9	Entero largo	209

Ejemplo

*Consulte el ejemplo del comando **SET PRINT MARKER**.*

GET PRINT OPTION

GET PRINT OPTION (opcion ; valor1 {; valor2})

Parámetro	Tipo		Descripción
opcion	Entero largo	➡	Número de opción
valor1	Entero largo, Texto	⬅	Valor 1 de la opción
valor2	Entero largo, Texto	⬅	Valor 2 de la opción

Descripción

*El comando **GET PRINT OPTION** devuelve los valores actuales de una opción de impresión.*

El parámetro opcion le permite especificar la opción a obtener. Puede obtener una opción estándar (entero largo), o un código de opción PDF (cadena). El comando devuelve, en los parámetros valor1 y (opcionalmente) valor2, el valor actual de la opción especificada.

*Para especificar una opción de impresión estándar, puede utilizar una de las siguientes constantes predefinidas, ubicadas en el tema "**Opciones de impresión**":*

Constante	Tipo	Valor	Comentario
Paper option	Entero largo	1	Si sólo utiliza valor1, que contiene el nombre del papel. Si se utilizan los dos parámetros, valor1 contiene el ancho del papel y valor2 contiene el alto del papel. El ancho y el alto se expresan en píxeles de pantalla. Utilice el comando **PRINT OPTION VALUES** para obtener el nombre, el alto y el ancho de todos los formatos de papel que ofrece la impresora.
Orientation option	Entero largo	2	valor1 únicamente: 1=Retrato, 2=Paisaje. Si se utiliza una opción de orientación diferente, **GET PRINT OPTION**devuelve 0 en valor1. **Versiones 64 bits**: esta opción se puede llamar desde una tarea de impresión, lo que significa que puede cambiar de vertical a horizontal, o viceversa, durante el mismo trabajo de impresión.
Scale option	Entero largo	3	valor1 únicamente: valor de la escala en porcentaje. Tenga cuidado, algunas impresoras no permiten modificar la escala. Si pasa un valor no válido, la propiedad se reinicia al 100% en el momento de la impresión.
Number of copies option	Entero largo	4	valor1 únicamente: número de copias a imprimir.
Paper source option	Entero largo	5	(Windows únicamente) valor1 únicamente: número correspondiente al índice, en el array de bandejas devuelto por el comando **PRINT OPTION VALUES**, de la bandeja de papel a utilizar. Esta opción sólo se puede utilizar en Windows.
Color option	Entero largo	8	(Windows únicamente) valor1 únicamente: código que especifica el modo para el manejo del color: 1=Blanco y negro (monocromo), 2=Color. **Versiones 64 bits**: esta opción no está disponible en versiones 4D de 64 bits (obsoleto)
Destination option	Entero largo	9	valor1: código que especifica el tipo de destino de la impresión: 1=Impresora, 2=Archivo (PC)/PS (Mac), 3=Archivo PDF, 5=Pantalla (opción del driver OS X). Si valor1 es diferente de 1 o 5, valor2 contiene un nombre de ruta para el documento resultante. Esta ruta se utilizará hasta que se especifique otra ruta. Si un archivo con el mismo nombre ya existe en el lugar de destino, será sustituido. Con **GET PRINT OPTION**, si el valor actual no está en la lista predefinida, valor1 contiene -1 y la variable sistema OK toma el valor 1. Si ocurre un error, valor1 y la variable sistema OK toman el valor 0. **Nota**: en Windows, puede definir el destino de impresión 3 (archivo PDF) cuando el driver PDF Creator ha sido instalado. Cuando se pasan los valores (9;3;ruta), 4D inicia automáticamente una impresión PDF "silenciosa" que tiene en cuenta los códigos de opción pasados (note que si pasa una cadena vacía en valor2 u omite este parámetro, aparece el diálogo de guardar archivo en el momento de la impresión.) Después de la impresión, los parámetros actuales se restauran. Esto simplifica el control de las impresiones PDF por 4D y permite escribir código multiplataforma. Cuando los valores (9;3;ruta) no se transmiten, la impresión no es controlada por 4D y los eventuales códigos de opciones de PDF Creator se ignoran.
Double sided option	Entero largo	11	(Windows únicamente) valor1: 0=Un solo lado o estándar, 1=Doble cara. Si valor1=1, valor2 contiene la unión: 0=Izquierda (valor predeterminado), 1=Unión superior. **Nota**: esta opción sólo se puede utilizar en Windows.
Spooler document name option	Entero largo	12	valor1 únicamente: nombre del documento de impresión actual, que aparece en la lista de documentos de la cola de impresión. El nombre definido para esta instrucción se utilizará para todos los documentos de impresión de la sesión hasta que un nuevo nombre o una cadena vacía no se pase. Para utilizar o restablecer el funcionamiento normal (usando el nombre del método en el caso de un método, el nombre de la tabla para un registro, etc.), pase una cadena vacía en valor1.
Mac spool file format option	Entero largo	13	(Mac únicamente) valor1 únicamente: 0=impresión en modo PDF (valor por defecto) 1 = impresión en modo PostScript. **Notas:** - Esta opción no tiene efecto en Windows.. - En OS X, la impresión se realiza en formato PDF de forma predeterminada. Sin embargo, el driver de impresión PDF no es compatible con imágenes PICT con información PostScript encapsulada, estas imágenes se generan, particularmente, por los softwares de dibujo vectorial. Para evitar este problema, esta opción le permite modificar el modo de impresión a utilizar en OS X para la sesión actual. Tenga en cuenta la impresión en modo PostScript puede conducir a efectos secundarios no deseados. **Versiones 64 bits:** esta opción no es compatible; Es reemplazada por la opción Driver PDF générique del comando **SET CURRENT PRINTER**.
Hide printing progress option	Entero largo	14	(Mac únicamente) valor1 únicamente: 1=ocultar ventanas de progreso, 0= mostrar las ventanas de progreso de impresión (por defecto). Esta es una opción particularmente útil en el caso de impresión PDF en OS X. **Nota**: ya existe una opción de progreso de impresión accesible vía el cuadro de diálogo Propiedades de la base (página Interfaz). Sin embargo, se aplica globalmente a la aplicación y no oculta todas las ventanas bajo OS X.
Page range option	Entero largo	15	valor1=primera página a imprimir (valor por defecto 1) y (opcional) valor2=número de la última página a imprimir (valor por defecto -1 = fin del documento).
Legacy printing layer option	Entero largo	16	(Versiones 4D 64 bits para Windows únicamente) valor1 únicamente: 1=seleccionar la antigua capa de impresión GDI para todos los trabajos de impresión subsiguientes. 0=seleccionar la capa de impresión D2D (por defecto). **Versiones 64 bits:** este selector sólo es compatible con las aplicaciones 4D 64 bits mono usuario en Windows; se ignora en otras plataformas. Está destinado principalmente para permitir plug-ins de antigua generación imprimir dentro de tareas de impresión 4D.

Un código de opción PDF consta de dos partes, TipoOpcion y NombreOpcion, combinadas como "TipoOpcion:NombreOpcion". Para mayor información sobre códigos de opción PDF y valores posibles, consulte la descripción del comando **SET PRINT OPTION**.

Nota: *el comando* ***GET PRINT OPTION*** *principalmente soporta impresoras PostScript. Puede utilizar este comando con otros tipos de impresoras, como PCL o tinta, pero en este caso, es posible que algunas opciones no estén disponibles.*

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si el comando ha sido ejecutado correctamente; de lo contrario, toma el valor 0.

Get print preview

Get print preview -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*True = Vista previa de impresión,* *False = No vista previa de impresión*

Descripción

*El comando **Get print preview** devuelve True si el comando **SET PRINT PREVIEW** se llamó con el valor **True** en el proceso actual.*

*Note que el usuario puede modificar esta opción antes de validar la caja de diálogo. Para obtener el modo final de impresión, debe utilizar el comando **Is in print preview**.*

GET PRINTABLE AREA

GET PRINTABLE AREA (altura {; largo})

Parámetro	Tipo		Descripción
altura	Entero largo	⇐	Altura del área de impresión
largo	Entero largo	⇐	Largo del área de impresión

Descripción

*El comando **GET PRINTABLE AREA** devuelve el tamaño en píxeles del área de impresión en los parámetros altura y largo el tamaño. Este tamaño depende de los parámetros de impresión actuales, la orientación del papel, etc.*

El tamaño devuelto no varía de una página a otra (después de un salto de página, por ejemplo).

*Asociado al comando **Get printed height**, este comando es útil para conocer el número de píxeles disponibles para la impresión o para centrar un objeto en la página.*

***Nota:** para mayor información sobre gestión de impresión y terminología en 4D, consulte la descripción del comando **GET PRINTABLE MARGIN**.*

Para saber el tamaño total de la página, puede:

- *añadir las márgenes ofrecidas por el comando **GET PRINTABLE MARGIN** a los valores devueltos por este comando.*
- *o utilizar la siguiente sintaxis:*

```
SET PRINTABLE MARGIN(0;0;0;0) ` Definir el margen del papel
GET PRINTABLE AREA(hPapel;wPapel) `Tamaño del papel
```

GET PRINTABLE MARGIN

GET PRINTABLE MARGIN (izquierda ; superior ; derecha ; inferior)

Parámetro	Tipo		Descripción
izquierda	Entero largo	⇐	Margen izquierda
superior	Entero largo	⇐	Margen superior
derecha	Entero largo	⇐	Margen derecha
inferior	Entero largo	⇐	Margen inferior

Descripción

*El comando **GET PRINTABLE MARGIN** devuelve los valores actuales de los diferentes márgenes definidos utilizando los comandos **Print form**, **PRINT SELECTION** y **PRINT RECORD**.*

Los valores son devueltos en píxeles con respecto al borde del papel.

*Es posible obtener el tamaño del papel como también calcular el área imprimible utilizando la función **GET PRINTABLE AREA** .*

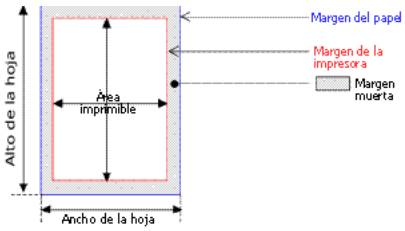
Gestión de márgenes de impresión

Por defecto, en 4D el cálculo de las impresiones se efectúa sobre la base de los "márgenes de la impresora". La ventaja de este sistema es que los formularios se adaptan automáticamente a las nuevas impresoras (ya que están situados en el área imprimible). Por otra parte, en el caso de los formularios pre-impresos, no era posible posicionar los elementos a imprimir de manera precisa porque un cambio de impresora podía modificar los márgenes de la impresora.

*Es posible basar la impresión de los formularios efectuados utilizando los comandos **Print form**, **PRINT SELECTION** y **PRINT RECORD** sobre un margen fijo el cual es idéntico en cada impresora: los márgenes del papel, es decir, los límites físicos de la hoja. Para hacer esto, simplemente utilice los comandos **GET PRINTABLE MARGIN**, **SET PRINTABLE MARGIN** y **GET PRINTABLE AREA**.*

Terminología de impresión

- ***Margen del papel**: el margen del papel corresponde a los límites físicos de la hoja.*
- ***Margen de impresión**: la margen de impresión es la margen más allá de la cual la impresora no puede imprimir (por razones físicas: rodillos de impresión, fin del recorrido del cabezal de impresión...). Varía de una impresora a otra y de un formato a otro.*
- ***Margen muerta:** esta es el área situada entre la margen del papel y el margen de la impresora.*



Get printed height

Get printed height -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Posición del marcador

Descripción

El comando ***Get printed height*** *devuelve la altura global (en píxeles) de la sección impresa por el comando* ***Print form****.*

El valor devuelto estará entre 0 (el borde superior de la página) y la altura total devuelta por el comando ***GET PRINTABLE AREA*** *(el tamaño máximo del área de impresión).*

Si imprime una nueva sección utilizando el comando ***Print form****, la altura de la nueva sección se añade a este valor. Si el área de impresión disponible es insuficiente para contener esta sección, se genera una nueva página y el valor devuelto es 0.*

Los márgenes de impresión derecha e izquierda no influyen en el valor devuelto, a diferencia de las márgenes inferior y superior (las cuales pueden definirse utilizando el comando ***SET PRINTABLE MARGIN****).*

Nota: *para mayor información sobre gestión de impresión y terminología en 4D, consulte la descripción del comando* ***GET PRINTABLE MARGIN****.*

Is in print preview

Is in print preview -> resultado

Parámetro	*Tipo*		*Descripción*
resultado	*Booleano*		*True = Vista previa,* *False = No vista previa*

Descripción

*El comando **Is in print preview** devuelve True si la opción **Vista previa de impresión** está seleccionada en la caja de diálogo de impresión y False de lo contrario. Esta configuración es local al proceso.*

*A diferencia del comando **Get print preview**, **Is in print preview** devuelve el valor final de la opción, después de la validación de la caja de diálogo por parte del usuario. Este comando permite determinar con certeza si la impresión toma lugar en modo "vista previa".*

Ejemplo

Este ejemplo permite tener en cuenta todos los tipos de impresiones:

```
SET PRINT PREVIEW(True) //Vista previa de impresión por defecto
PRINT SETTINGS
If(OK=1)
 //El usuario puede haber cambiado el destino de impresión
   If(Is in print preview) // True si vista previa
      FORM SET OUTPUT([Facturas] ;"toScreen")
   Else
      FORM SET OUTPUT([Facturas] ;"toPrinter"
   End if
   OPEN PRINTING JOB
   ALL RECORDS([Facturas])
   PRINT SELECTION([Facturas];>)
   CLOSE PRINTING JOB
End if
```

Level

Level -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Nivel de ruptura o del encabezado actual

Descripción

Level *se utiliza para determinar el nivel de ruptura o del encabezado actual. Devuelve el nivel de ruptura durante los eventos* ***On Header*** *y* ***On Printing Break****.*

El nivel 0 es el último nivel a imprimir y es apropiado para la impresión de un total general. ***Level*** *devuelve 1 cuando 4D imprime una ruptura en el primer campo de ordenación, 2 cuando 4D imprime una ruptura en el segundo campo ordenado, y así sucesivamente.*

Ejemplo

Este ejemplo es una plantilla para un método de formulario. Muestra cada evento posible mientras un informe utiliza un formulario como formulario de salida. ***Level*** *se llama cuando un encabezado o una ruptura se están imprimiendo:*

```
  ` Método de formulario para un formulario de salida utilizado por un informe
$vpFormTable:=Current form table
Case of
  ` ...
   :(Form event=On Header)
  ` Se va a imprimir el área de encabezado
      Case of
         :(Before selection($vpFormTabla->))
  ` El código para la primera ruptura del encabezado debe ir acá
         :(Level=1)
  ` El código para la ruptura del encabezado nivel 1 debe ir acá
         :(Level=2)
  ` El código para la ruptura del encabezado nivel 2 debe ir acá
  ` ...
      End case
   :(Form event=On Printing Details)
  ` Se va a imprimir un registro
  ` El código para cada registro va acá
   :(Form event=On Printing Break)
  ` Se va a imprimir un área de ruptura
      Case of
         :(Level=0)
  ` El código para la ruptura 0 va acá
         :(Level=1)
  ` El código para la ruptura 1 va acá
  ` ...
      End case
   :(Form event=On Printing Footer)
      If(End selection($vpFormTabla->))
  ` El código para el último pie de página debe ir acá
      Else
  ` El código para un pie de página deber ir acá
      End if
End case
```

OPEN PRINTING JOB

OPEN PRINTING JOB
Este comando no requiere parámetros

Descripción

*El comando **OPEN PRINTING JOB** abre una tarea de impresión y apila todas las órdenes de impresión ejecutadas hasta que se llame el comando **CLOSE PRINTING JOB**. Este comando le permite controlar los trabajos de impresión y, más particularmente, asegurar que ninguna tarea de impresión "parásita" pueda ser insertada en una secuencia de impresión.*

*El comando **OPEN PRINTING JOB** puede utilizarse con todos los comandos de impresión 4D, los comandos del editor de informes rápidos, y los comandos de impresión de los plug-ins 4D Write y 4D View. Por otra parte, este comando no es compatible con los plug-ins 4D Chart y 4D Draw, así como la mayoría de los plug-ins de terceras partes.*

Cuando un trabajo de impresión se abre con este comando, la impresora se coloca en modo "ocupado" hasta que la impresión se lance efectivamente. Si un plug-in no compatible lanza una impresión en este contexto, se devuelve el error "impresora ocupada".

*Debe llamar al comando **CLOSE PRINTING JOB** para determinar el trabajo de impresión y enviar el documento de impresión a la impresora. Si omite este comando, el documento de impresión permanecerá en la pila y la impresora no estará disponible hasta que salga de la aplicación 4D.*

El trabajo de impresión es local al proceso pero la impresión se realiza a nivel global, un sólo trabajo de impresión puede abrirse a la vez en 4D, todos los procesos combinados.

***OPEN PRINTING JOB** utiliza los parámetros de impresión actuales (parámetros por defecto o definidos vía los comandos **PAGE SETUP** y/o **SET PRINT OPTION**). Los comandos que modifican los parámetros de impresión deben ser llamados antes de **OPEN PRINTING JOB**. De lo contrario, se genera un error.*

PAGE BREAK

PAGE BREAK {(* | >)}

Parámetro	Tipo	Descripción
* | >	➡	* Cancela la impresión iniciada por Print form, o > Manda un trabajo de impresión

Descripción

PAGE BREAK *dispara la impresión de datos enviados a la impresora y provocar un salto de página.* ***PAGE BREAK*** *se utiliza conjuntamente con form (en el contexto del evento de formulario* ***On Printing Detail****) para forzar saltos de página e imprimir la última página creada en memoria. No utilice* ***PAGE BREAK*** *con el comando* ***PRINT SELECTION****. Es mejor utilizar* ***Subtotal*** *o* ***BREAK LEVEL*** *con el parámetro opcional para generar saltos de página.*

*Los parámetros * y > son opcionales.*

*El parámetro * le permite cancelar un trabajo de impresión iniciado por el comando* ***Print form****. La ejecución de este comando detiene inmediatamente los trabajos de impresión en progreso.*

Nota: *bajo Windows, este mecanismo puede ser trastornado por las propiedades de cola de espera del servidor de impresión. Si la impresora se configura para comenzar la impresión de inmediatamente, la cancelación no será efectiva. Para que el comando* ***PAGE BREAK(*)*** *funcione correctamente, es preferible elegir la propiedad "Iniciar impresión cuando la última página haya entrado en la cola" de la impresora.*

El parámetro > modifica el funcionamiento del comando ***PAGE BREAK****. Esta sintaxis tiene dos efectos:*

- *Mantiene la impresión abierta hasta que el comando* ***PAGE BREAK*** *se ejecuta nuevamente sin un parámetro.*
- *Da prioridad al trabajo de impresión. Ninguna otra impresión puede llevarse a cabo hasta que el trabajo de impresión haya terminado.*
  La segunda opción es particularmente útil cuando se utiliza con un trabajo de impresión en cola. El parámetro > garantiza que el trabajo de impresión será realizado a partir de un solo archivo. Esto reducirá el tiempo de impresión.

Nota: *durante una impresión de pantalla, si el usuario hace clic en Cancelar en la caja de diálogo de previsualización, el comando* ***PAGE BREAK*** *define la variable sistema OK en 0.*

Ejemplo 1

Ver el ejemplo del comando ***Print form****.*

Ejemplo 2

Consulte el ejemplo del comando ***SET PRINT MARKER****.*

PAGE SETUP

PAGE SETUP ({tabla ;} formulario)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a la que pertenece el formulario, o Tabla por defecto, si se omite
formulario	Cadena	➡	Formulario a utilizar para definir los parámetros de impresión

Descripción

PAGE SETUP define la configuración de la página para la impresora almacenada con formulario. Los parámetros de impresión son almacenados con el formulario cuando el formulario se guarda en el entorno Diseño.

En los siguientes tres casos, las cajas de diálogo de impresión no se muestran y la impresión se realiza con los parámetros de impresión por defecto:

- Una llamada a **PRINT SELECTION** a la cual pasa el parámetro opcional *,
- Una llamada a **PRINT RECORD** a la cual pasa el parámetro opcional *,
- Una serie de llamadas a **PRINT FORM** no precedida por una llamada a **PRINT SETTINGS**.

Llamar **PAGE SETUP** le permite, en este caso, saltar las cajas de diálogo de impresión Y utilizar los parámetros de impresión diferentes a los parámetros por defecto.

Ejemplo

Varios formularios (vacíos) son creados por una tabla llamada [Dibujos]. El formulario "PS100" es asignado a la configuración de la página con una escala de 100%, el formulario "PS90" se asigna a una configuración de página con escala de 90%, y así sucesivamente. El siguiente método de proyecto le permite imprimir la selección de una tabla utilizando varias escalas sin tener que especificar la escala en las cajas de diálogo de impresión (las cuales no aparecen), cada vez:

```
  ` Método de proyecto IMPRESION ESCALADA AUTOMATICA
  ` IMPRESION ESCALADA AUTOMATICA ( Puntero; Cadena {; Long } )
  ` IMPRESION ESCALADA AUTOMATICA ( ->[Tabla]; "FormularioSalida" {; Escala } )
If(Count parameters>=3)
   PAGE SETUP([Dibujos];"PS"+String($3))
   If(Count parameters>=2)
      OUTPUT FORM($1->;$2)
   End if
End if
If(Count parameters>=1)
   PRINT SELECTION($1->;*)
Else
   PRINT SELECTION(*)
End if
```

Una vez este método de proyecto está escrito, puede llamarlo de esta forma:

```
  ` Buscar facturas actuales
QUERY([Facturas];[Facturas]Pagadas=False)
  ` Impresión de un informe reducido al 90%
IMPRESION ESCALADA AUTOMATICA(->[Facturas];"Informe resumen";90)
  ` Impresión de un informe reducido al 50%
IMPRESION ESCALADA AUTOMATICA(->[Facturas];"Informe detallado";50)
```

Print form

Print form ({tabla ;} formulario {; area1 {; area2}}) -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a imprimir, o Tabla por defecto, si se omite
formulario	Cadena, Objeto	➡	Formulario a imprimir
area1	Entero largo	➡	Marcador de impresión, o Área de inicio (si area2 se especifica)
area2	Entero largo	➡	Área de fin (si área1 se especifica)
Resultado	Entero largo	↩	Altura de la sección impresa

Descripción

Print form simplemente imprime formulario con los valores actuales de los campos y variables. Generalmente este comando se utiliza para imprimir informes muy complejos que necesiten un control total del proceso de impresión. **Print form** no procesa registros, ni rupturas o saltos de páginas. Estas operaciones son su responsabilidad. **Print form** imprime campos y variables en un marco de tamaño fijo únicamente.

En el parámetro formulario, puede pasar:

- el nombre de un formulario, o
- la ruta (en sintaxis POSIX) a un archivo .json válido que contiene una descripción del formulario a usar (ver **Ruta de archivo del formulario**), o
- un objeto que contiene una descripción del formulario.

Como **Print form** no genera un salto de página después de imprimir el formulario, es fácil combinar diferentes formularios en la misma página. Entonces, **Print form** es ideal para efectuar tareas de impresión complejas que involucren diferentes tablas y diferentes formularios. Para forzar un salto de página entre formularios, utilice el comando **PAGE BREAK**. Para pasar a la siguiente página de un formulario cuya altura es mayor que el espacio disponible, llame el comando **CANCEL** antes del comando **PAGE BREAK**.

Se pueden utilizar tres sintaxis diferentes:

- **Impresión del área de detalle**

Sintaxis:

```
altura:=Print form(miTabla;miForm)
```

En este caso, **Print form** sólo imprime el área de detalle (el área entre la línea encabezado y la línea detalle) del formulario.

- **Impresión del área del formulario**

Sintaxis:

```
altura:=Print form(miTabla;miForm;marcador)
```

En este caso, el comando imprimirá la sección designada por el marcador. Pase en el parámetro marcador una de las constantes del tema **Área de formulario** :

Constante	Tipo	Valor
Form break0	Entero largo	300
Form break1	Entero largo	301
Form break2	Entero largo	302
Form break3	Entero largo	303
Form break4	Entero largo	304
Form break5	Entero largo	305
Form break6	Entero largo	306
Form break7	Entero largo	307
Form break8	Entero largo	308
Form break9	Entero largo	309
Form detail	Entero largo	0
Form footer	Entero largo	100
Form header	Entero largo	200
Form header1	Entero largo	201
Form header10	Entero largo	210
Form header2	Entero largo	202
Form header3	Entero largo	203
Form header4	Entero largo	204
Form header5	Entero largo	205
Form header6	Entero largo	206
Form header7	Entero largo	207
Form header8	Entero largo	208
Form header9	Entero largo	209

• *Impresión de sección*

Sintaxis:

```
altura:=Print form(miTabla;miForm;arealnicio;areaFin)
```

En este caso, el comando imprimirá la sección incluida entre los parámetros areaInicio y areaFin Parámetros. Los valores introducidos deben expresarse en píxeles.

*El valor devuelto por **Print form** indica la altura del área de impresión. Este valor será tomado en cuenta automáticamente por el comando **Get printed height**.*

*Las cajas de diálogo de impresión no aparecen cuando utiliza **Print form**. El informe no utiliza los parámetros de impresión definidos para el formulario en el entorno Diseño. Hay dos formas de especificar los parámetros de impresión antes de efectuar una serie de llamadas a **Print form**:*

- *Llamar **PRINT SETTINGS**. En este caso, usted le permite al usuario elegir los parámetros.*
- *Llamar **PAGE SETUP**. En este caso, los parámetros de impresión se especifican por programación.*

***Print form** construye cada página impresa en memoria. Cada página se imprime cuando la página en memoria está llena o cuando usted llama a **PAGE BREAK**. Para asegurar la impresión de la última página después de utilizar **Print form**, debe concluir con el comando **PAGE BREAK** (excepto en el contexto de un **OPEN PRINTING JOB**, ver nota). De lo contrario, si la última página no está llena, permanece en memoria y no se imprime.*

***Atención:** si el comando se llama en el contexto de un trabajo de impresión abierto con **OPEN PRINTING JOB**, NO debe llamar a **PAGE BREAK** para la última página porque se imprime automáticamente por el comando **CLOSE PRINTING JOB**. Si llama **PAGE BREAK** en este caso, se imprime una página vacía.*

Este comando imprime las áreas y objetos externos (por ejemplo, las áreas 4D Write o 4D View). El área se reinicializa para cada ejecución del comando.

***Atención:** **Print form** no imprime subformularios. Para imprimir sólo un formulario con tales objetos, utilice mejor **PRINT RECORD**.*

***Print form** genera sólo un evento On Printing Detail para el método formulario.*

***4D Server:** este comando puede ejecutarse en 4D Server dentro del framework de un procedimiento almacenado. En este contexto:*

- *Asegúrese de que no aparezca ninguna caja de diálogo en el equipo servidor (excepto para un requerimiento específico).*
- *En el caso de un problema relacionado con la impresora (sin papel, impresora desconectada, etc.), no se genera un mensaje de error.*

Ejemplo 1

***Print form** no imprime subformularios. Para imprimir sólo un formulario con tales objetos, utilice mejor **PRINT RECORD**.*

***Print form** genera únicamente un evento **On Printing Detail** por método de formulario.*

***4D Server:** este comando puede ejecutarse en 4D Server en el marco de un procedimiento almacenado. En este contexto:*

- *Asegúrese de que no aparezca ninguna caja de diálogo en el equipo servidor (excepto para una necesidad específica).*
- *En el caso de un problema relacionado con la impresora (sin papel, impresora desconectada, etc.), no se genera un mensaje de error.*

Ejemplo 2

*El siguiente ejemplo funciona como lo haría un comando **PRINT SELECTION**. Sin embargo, el informe utiliza uno de los dos formularios diferentes, dependiendo de si el registro es para un cheque o para un depósito:*

```
QUERY([Registro]) ` Select the records
If(OK=1)
   ORDER BY([Registro]) ` Ordenar los registros
   If(OK=1)
      PRINT SETTINGS ` Mostrar las cajas de diálogo de impresión
      If(OK=1)
         For($vlRegistro;1;Records in selection([Registro]))
            If([Registro]Tipo ="Cheque")
               Print form([Registro];"SalidaCheque") ` Utilice un formulario de cheques
            Else
               Print form([Registro];"SalidaDeposito") ` Utilizar otro formulario de depósitos
            End if
            NEXT RECORD([Registro])
         End for
         PAGE BREAK ` Asegúrese de que la última página se imprima
      End if
   End if
End if
```

PRINT LABEL

PRINT LABEL ({tabla }{;}{ doc {; * | >}})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a imprimir, o Tabla por defecto, si se omite
doc	Cadena	➡	Nombre del documento de etiquetas del disco
* | >		➡	* para suprimir las cajas de diálogo de impresión, o > para no reiniciar los parámetros de impresión

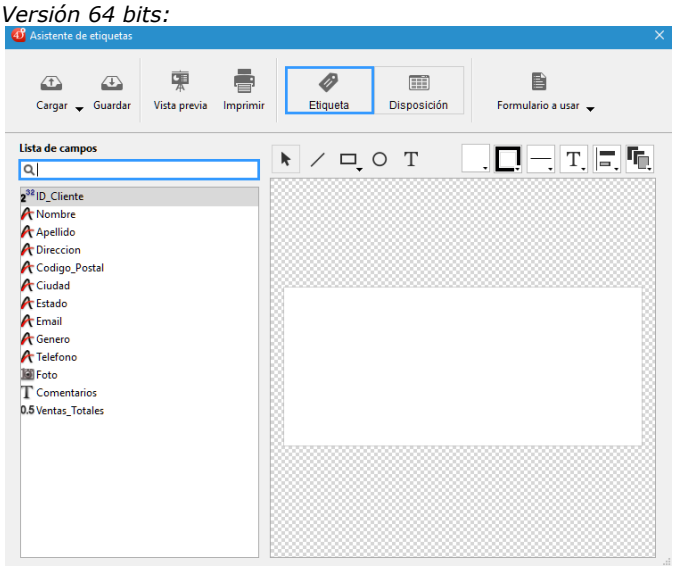
Descripción

PRINT LABEL *le permite imprimir etiquetas con los datos de la selección de tabla.*

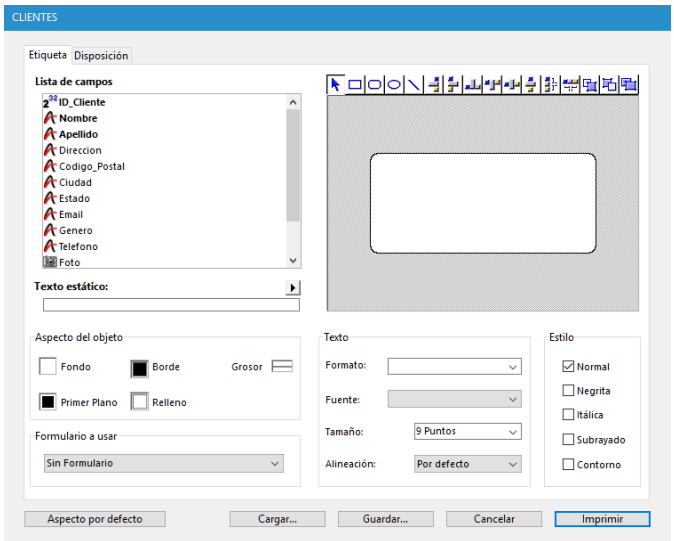
Si no especifica el parámetro documento, ***PRINT LABEL*** *imprime la selección actual de tabla como etiquetas, utilizando el formulario de salida actual. No puede utilizar este comando para imprimir subformularios. Para mayor información sobre la creación de formularios para etiquetas, consulte el Manual de Diseño.*

Si especifica el parámetro documento, ***PRINT LABEL*** *le permite tener acceso al Asistente de etiquetas (mostrado a continuación) o imprimir un documento de etiquetas existente almacenado en el disco. Ver el ejemplo a continuación.*

Versión 64 bits:



Versión 32 bits:



Nota de compatibilidad*: la versión 32 bits del editor de etiquetas sólo admite caracteres ASCII. Si necesita usar un conjunto de caracteres más extendido, debe imprimir etiquetas usando el formulario de salida actual.*

Por defecto, ***PRINT LABEL*** *muestra las dos cajas de diálogo de parámetros de impresión (4D versión 32 bits) o la caja de diálogo de impresión (4D versión 64 bits). Si el usuario cancela una de las cajas de diálogo de impresión, el comando se cancela y las etiquetas no se imprimen.*

Puede suprimir estas cajas de diálogo utilizando el parámetro opcional asterisco (*) o el parámetro opcional "mayor que" (>):

- *El parámetro * causa una impresión con los parámetros de impresión actuales.*
- *Además, el parámetro > provoca un trabajo de impresión sin reinicializar los parámetros de impresión actuales. Este parámetro es útil para ejecutar varias llamadas sucesivas a* ***PRINT LABEL*** *(por ejemplo al interior de un bucle) mientras mantiene los parámetros de impresión personalizados previamente definidos. Para ver un ejemplo sobre el uso de este parámetro, consulte la descripción del comando* ***PRINT RECORD****.*

Note que este parámetro no tiene efecto si se utiliza el asistente de creación de etiquetas.

*Si no se utiliza el asistente de creación de etiquetas, la variable sistema OK toma el valor 1 si todas las etiquetas se imprimen; de lo contrario, toma el valor 0 (cero) (por ejemplo, si el usuario hizo clic en el botón **Cancelar** en las cajas de diálogo de impresión).*

*Si especifica el parámetro documento, las etiquetas se imprimen con los parámetros definidos en documento. Si documento es una cadena vacía (""), **PRINT LABEL** presentará una caja de diálogo estándar de apertura de documentos, permitiendo al usuario seleccionar el archivo de etiquetas a utilizar. Si documento es el nombre de un documento que no existe (por ejemplo, si pasa char(1) en documento), el asistente de creación de etiquetas aparece, permitiendo al usuario definir su formato de etiquetas.*

***Nota:** si la tabla fue declarada "invisible" en el entorno Diseño, no se mostrará el asistente de etiquetas.*

***4D Server:** este comando puede ejecutarse en 4D Server en el marco de un procedimiento almacenado. En este contexto:*

- *Asegúrese de que ninguna caja de diálogo aparezca en el equipo servidor (excepto por una necesidad específica). Para hacer esto, es necesario llamar al comando con el parámetro * o > .*
- *La sintaxis que hace que el editor de etiquetas aparezca no funciona con 4D Server; en este caso, la variable sistema OK toma el valor 0.*
- *En el caso de un problema relacionado de impresora (sin papel, impresora desconectada, etc.), no se genera un mensaje de error.*

Ejemplo 1

El siguiente ejemplo imprime las etiquetas utilizando el formulario de salida de una tabla. El ejemplo utiliza dos métodos. El primero es un método de proyecto que designa el formulario de salida a utilizar y luego imprime las etiquetas:

```
ALL RECORDS([Direcciones]) ` Selección de todos los registros
FORM SET OUTPUT([Direcciones];"Imprimir Etiqueta") ` Selección del formulario de salida
PRINT LABEL([Direcciones]) ` Impresión de etiquetas
FORM SET OUTPUT([Direcciones];"Salida") ` Restablecimiento del formulario de salida por defecto
```

El segundo método es el método de formulario del formulario "Imprimir Etiqueta". El formulario contiene una variable llamada vEtiq, que se utiliza para mantener los campos concatenados. Si el segundo campo de direcciones (Dir2) está vacío, es eliminado por el método. Note que esta operación es realizada automáticamente por el asistente de creación de etiquetas. El método de formulario crea la etiqueta para cada registro:

```
`Método de formulario [Direcciones]; "Etiqueta salida"
Case of
   :(Form event=On Load)
      vEtiq:=[Direcciones]Nom1+" "+[Direcciones]Nom2+Char(13)+[Direcciones]Dir1+Char(13)
      If([Direcciones]Dir2#"")
         vEtiq:=vLabel+[Direcciones]Dir2+Char(13)
      End if
      vEtiq:=vEtiq+[Direcciones]Ciudad+", "+[Direcciones]Estado+" "+[Direcciones]CodigoPostal
End case
```

Ejemplo 2

El siguiente ejemplo le permite al usuario efectuar una búsqueda en la tabla [Personas], y luego imprime automáticamente las etiquetas "Mis etiquetas":

```
QUERY([Personas])
If(OK=1)
   PRINT LABEL([Personas];"Mis etiquetas";*)
End if
```

Ejemplo 3

El siguiente ejemplo le permite al usuario efectuar una búsqueda en la tabla [Personas], y después le permite al usuario elegir las etiquetas a imprimir:

```
QUERY([Personas])
If(OK=1)
   PRINT LABEL([Personas];"")
End if
```

Ejemplo 4

El siguiente ejemplo le permite al usuario efectuar una búsqueda en la tabla [Personas] y luego muestra el Asistente de etiquetas de manera que el usuario pueda diseñar, guardar, cargar e imprimir todo tipo de etiquetas:

```
QUERY([Personas])
If(OK=1)
```

```
    PRINT LABEL([Personas];Char(1))
End if
```

Print object

Print object ({* ;} objeto {; posX {; posY {; ancho {; alto}}}}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de ojbeto (cadena). Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o variable (si se omite *)
posX	Entero largo	➡	Ubicación horizontal del objeto
posY	Entero largo	➡	Ubicación vertical del objeto
ancho	Entero largo	➡	Ancho del objeto (píxeles)
alto	Entero largo	➡	Alto del objeto (píxeles)
Resultado	Booleano	⬅	True = objeto impreso completamente, de lo contrario False

Descripción

*El comando **Print object** permite imprimir el o los objetos de formulario designado(s) por los parámetros objeto y *, en la ubicación definida por los parámetros posX y posY.*

*Antes de llamar el comando **Print object**, debe designar el formulario tabla o proyecto que contiene los objetos a imprimir, utilizando el comando **FORM LOAD**.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena de caracteres). Si no pasa el parámetro *, está indicando que objeto es una variable. En este caso, se pasa una referencia de variable (tipo de objeto únicamente) en lugar de una cadena.*

Los parámetros posX y posY especifican el punto de partida para la impresión del o de los objeto(s). Estos valores deben expresarse en píxeles. Si estos parámetros se omiten, el objeto se imprime de acuerdo a su ubicación en el formulario.

*Los parámetros ancho y alto se utilizan para especificar el ancho y el alto del objeto de formulario. El comando **Print object** no trata objetos de tamaño variable. Debe utilizar el comando **OBJECT GET BEST SIZE** para manejar el tamaño de los objetos. También puede utilizar el comando **OBJECT GET BEST SIZE** para determinar el tamaño más adecuado para los objetos que contienen texto. Del mismo modo, **Print object** no provoca saltos de página automáticos. Debe manejarlos de acuerdo a sus necesidades.*

Puede utilizar los comandos de 4D para modificar rápidamente las propiedades del objeto (color, tamaño, etc).

*El comando devuelve True si el objeto se ha impreso completamente y False si este no es el caso, en otras palabras, si no pudo imprimir todos los datos asociados con el objeto dentro del marco establecido. Normalmente, el comando devuelve False cuando se imprime un list box si todas las líneas del list box no se pudieron imprimir. En este caso, basta con llamar al comando **Print object** varias veces hasta que devuelva True: un mecanismo específico provoca de forma automática el desplazamiento del contenido del objeto después de cada llamada.*

***Notas**:*

- *En la versión actual de 4D, sólo los objetos de tipo list box tienen este mecanismo (el comando siempre devuelve True para cualquier otro tipo de objeto). En las próximas versiones de 4D, este funcionamiento se ampliará a otros objetos con contenidos variables.*
- *El comando **LISTBOX GET PRINT INFORMATION** permite controlar el estado de la impresión durante la operación.*

*El comando **Print object** sólo puede ser utilizado en el contexto de un trabajo de impresión abierto previamente con el comando **OPEN PRINTING JOB**. Si no se llama en este contexto, el comando no hace nada. Varios comandos **Print object** pueden llamarse en el mismo trabajo de impresión.*

***Nota**: las listas jerárquicas, los subformularios y las áreas web no se pueden imprimir.*

Ejemplo 1

Ejemplo de impresión de diez objetos en un formulario:

```
PRINT SETTINGS
If(OK=1)
   OPEN PRINTING JOB
   If(OK=1)
      FORM LOAD("PrintForm")
      x:=100
      y:=50
      GET PRINTABLE AREA(hpaper;wpaper)
      For($i;1;10)
         OBJECT GET BEST SIZE(*;"Obj"+String($i);bestwidth;bestheight)
         $end:=Print object(*;"Obj"+String($i))
         y:=y+bestheight+15
         If(y>hpaper)
            PAGE BREAK(>)
            y:=50
```

```
            End if
        End for
    End if
    CLOSE PRINTING JOB
End if
```

Ejemplo 2

Ejemplo de impresión de un list box completo:

```
Repeat
    $end:=Print object(*;"mylistbox")
Until($end)
```

PRINT OPTION VALUES

PRINT OPTION VALUES (opción ; arrayNoms {; info1Array {; info2Array}})

Parámetro	Tipo		Descripción
opción	Entero largo	➡	Número de opción
arrayNoms	Array texto	⬅	Nombres de los valores
info1Array	Array entero largo	⬅	Valores (1) de la opción
info2Array	Array entero largo	⬅	Valores (2) de la opción

Descripción

En nomsArray, el comando **PRINT OPTION VALUES** devuelve una lista de nombres de valores disponibles para la opcion de impresión definida. Opcionalmente, puede recuperar la información para cada valor en info1Array y info2Array.

El parámetro opcion le permite especificar la opción a obtener. Debe pasar una de las siguientes constantes del tema "**Opciones de impresión**" (opciones que pueden devolver las listas de nombres de valores):

Constante	Tipo	Valor	Comentario
Paper option	Entero largo	1	Si sólo utiliza valor1, que contiene el nombre del papel. Si se utilizan los dos parámetros, valor1 contiene el ancho del papel y valor2 contiene el alto del papel. El ancho y el alto se expresan en píxeles de pantalla. Utilice el comando **PRINT OPTION VALUES** para obtener el nombre, el alto y el ancho de todos los formatos de papel que ofrece la impresora.
Paper source option	Entero largo	5	(Windows únicamente) valor1 únicamente: número correspondiente al índice, en el array de bandejas devuelto por el comando **PRINT OPTION VALUES**, de la bandeja de papel a utilizar. Esta opción sólo se puede utilizar en Windows.

Después de la ejecución del comando, el array nomsArray así como donde aplique los arrays info1Array e info2Array serán llenados por el comando con los nombres e información de los valores disponibles.

Si pasa el valor 1 (paper option) en el parámetro opcion, el comando devolverá la siguiente información:

- en nomsArray, los nombres de los formatos de papel disponibles;
- en info1Array, las alturas de cada formato de papel;
- en info2Array, los largos de cada formato de papel.

Nota: para obtener esta información, el driver de impresión debe tener acceso a un archivo de descripción (PPD) válido de la impresora.

Para obtener un formato de papel específico utilizando el comando **SET PRINT OPTION**, puede pasar uno de los valores de nomsArray, los valores correspondientes de info1Array e info2Array.

Si pasa el valor 5 (paper source option) en el parámetro opcion, el comando devuelve los nombres de las diferentes bandejas disponibles en nomsArray y sus números Windows ID internos en info1Array (info2Array permanece vacío).

El orden de los valores en los arrays está definido por el driver de impresión. Para indicar una bandeja utilizando el comando **SET PRINT OPTION**, debe pasar el índice del elemento que desea en el array nomsArray o info1Array.

Nota: esta opción sólo puede ser utilizada bajo Windows.

Para mayor información sobre las diferentes opciones de impresión, consulte la descripción de **SET PRINT OPTION** y **GET PRINT OPTION**.

Toda la información devuelta por estos comandos es suministrada por el sistema operativo. Consulte la documentación de su sistema para más detalles sobre opciones específicas.

Nota: el comando **PRINT OPTION VALUES** sólo funciona con impresoras PostScript.

PRINT RECORD

PRINT RECORD ({tabla}{;}{* | >})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual imprimir el registro actual o Tabla por defecto, si se omite
* | >	Operador	➡	* para suprimir las cajas de diálogo de impresión, o > para no reiniciar los parámetros de impresión

Descripción

PRINT RECORD *imprime el registro actual de tabla, sin modificar la selección actual. El formulario de salida actual se utiliza para la impresión. Si no hay registro actual para tabla,* ***PRINT RECORD*** *no hace nada.*

Puede imprimir subformularios y objetos externos con el comando ***PRINT RECORD****. Esto no es posible con* ***Print form****.*

Nota: *si hay modificaciones en el registro que no han sido guardadas, este comando imprime los valores de los campos modificados, no los valores en disco.*

Por defecto, ***PRINT RECORD*** *muestra las dos cajas de diálogo de parámetros de impresión (4D versión 32 bits) o la caja de diálogo de impresión (4D versión 64 bits). Si el usuario cancela una de las cajas de diálogo de impresión, el comando se cancela y no se imprime el registro.*
Puede suprimir estas cajas de diálogo utilizando el parámetro opcional asterisco (*) o el parámetro opcional "mayor que" (>):

- *El parámetro * produce un trabajo de impresión utilizando los parámetros de impresión actual (parámetros por defecto o definidos por los comandos* ***PAGE SETUP*** *y/o* ***SET PRINT OPTION****).*
- *Además, el parámetro > produce un trabajo de impresión sin reinicializar los parámetros de impresión actual. Este parámetro es útil para ejecutar varias llamadas consecutivas a* ***PRINT RECORD*** *(por ejemplo al interior de un bucle) mientras mantiene los parámetros de impresión personalizados definidos previamente.*

4D Server: *este comando puede ejecutarse en 4D Server dentro del marco de un procedimiento almacenado. En este contexto:*

- *Asegúrese que ninguna caja de diálogo aparezca en el equipo servidor (excepto para una necesidad específica). Para hacer esto, es necesario llamar al comando con el parámetro * o >.*
- *En caso de un problema con la impresora (sin papel, impresora desconectada, etc.), no se genera mensaje de error.*

Ejemplo 1

El siguiente ejemplo imprime el registro actual de la tabla [Facturas]. El código está en el método de objeto de un botón ***Imprimir*** *en el formulario de entrada. Cuando el usuario hace clic en el botón, el registro se imprime utilizando un formulario de salida diseñado para este propósito.*

```
FORM SET OUTPUT([Facturas];"Print One From Data Entry") ` Selección del formulario para impresión
PRINT RECORD([Facturas];*) ` Imprimir las facturas(sin mostrar diálogos de impresión)
FORM SET OUTPUT([Facturas];"Standard Output") ` Restauración del formulario de salida anterior
```

Ejemplo 2

El siguiente ejemplo imprime el mismo registro actual en dos formularios diferentes. El código está en el método de objeto de un botón ***Imprimir*** *en el formulario de entrada. Usted quiere definir parámetros de impresión personalizados y luego utilizarlos en dos formularios.*

```
PRINT SETTINGS `El usuario define los parámetros de impresión
If(OK=1)
   FORM SET OUTPUT([Empleados];"Detallado") `Usar el primer formulario de impresión
   PRINT RECORD([Empleados];>) `Imprimir utilizando los parámetros definidos por el usuario
   FORM SET OUTPUT([Empleados];"Simple") `Usar el segundo formulario de impresión
   PRINT RECORD([Empleados];>) `Imprimir utilizando los parámetros definidos por el usuario
   FORM SET OUTPUT([Empleados];"Output") `Restaurar el formulario de salida por defecto
End if
```

PRINT SELECTION

PRINT SELECTION ({tabla}{;}{* | >})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual imprimir la selección, o Tabla por defecto, si se omite
* | >	Operador	➡	* para eliminar las cajas de diálogo de impresión, o > para no reiniciar los parámetros de impresión

Descripción

PRINT SELECTION imprime la selección actual de tabla. Los registros se imprimen en el formulario de salida actual de la tabla del proceso actual. PRINT SELECTION realiza la misma acción que el comando Imprimir del entorno Usuario. Si la selección está vacía, PRINT SELECTION no hace nada.

Por defecto, PRINT SELECTION muestra las dos cajas de diálogo de parámetros de impresión (4D versión 64 bits) o la caja de diálogo de impresión (4D versión 64 bits). Si el usuario cancela una de las cajas de diálogo de impresión, el comando se cancela y el informe no se imprime.

Puede eliminar estas cajas de diálogo utilizando el parámetro opcional asterisco (*) o el parámetro opcional "mayor que" (>):
*• El parámetro * provoca un trabajo de impresión utilizando los parámetros de impresión actuales (parámetros por defecto o aquellos definidos por los comandos **PAGE SETUP** y/o **SET PRINT OPTION**).*
*• Además, el parámetro > provoca un trabajo de impresión sin reinicializar los parámetros de impresión actuales. Este parámetro es útil para ejecutar varias llamadas sucesivas a PRINT SELECTION (por ejemplo al interior de un bucle) conservando los parámetros de impresión personalizados previamente definidos. Para ver un ejemplo de utilización de este parámetro, consulte la descripción del comando **PRINT RECORD**.*

Durante la impresión, el método de formulario de salida y los métodos de objeto del formulario se ejecutan en función de los eventos seleccionados en las propiedades de los formularios y de los objetos, en el entorno Diseño, así como de los eventos generados efectivamente:

- *Un evento On Header se genera justo antes que el área de encabezado se imprima.*
- *Un evento On Printing Detail se genera justo antes que un registro se imprima.*
- *Un evento On Printing Break se genera justo antes que un área de ruptura se imprima.*
- *Un evento On Printing Footer se genera justo antes que un pie de página se imprima.*

*Puede saber si PRINT SELECTION está imprimiendo el primer encabezado probando **Before selection** durante un evento On Header. Igualmente puede verificar el último pie de página, probando **End selection** durante un evento **On Printing Footer**. Para mayor información, consulte la descripción de estos comandos, como también de los comandos **Evento formulario** y **Level**.*

*Para imprimir una selección ordenada con subtotales o rupturas utilizando PRINT SELECTION, debe primero ordenar la selección. Luego, en cada área de ruptura del informe, incluir una variable con un método de objeto que asigne el subtotal a la variable. Igualmente puede utilizar funciones estadísticas y aritméticas como **Sum** y **Average** para asignar valores a las variables. Para mayor información, consulte las descripciones de **Subtotal**, **BREAK LEVEL** y **ACCUMULATE**.*

***Advertencia:** no utilice el comando **PAGE BREAK** con el comando PRINT SELECTION. **PAGE BREAK** está reservado para ser utilizado con el comando **Print form**.*

Después de un llamado a PRINT SELECTION, la variable OK toma el valor 1 si la impresión se ha completado. Si la impresión fue interrumpida, la variable OK toma el valor 0 (cero) (por ejemplo si el usuario hizo clic en Cancelar en las cajas de diálogo de impresión).

***4D Server:** este comando puede ejecutarse en 4D Server en el marco de un procedimiento almacenado. En este contexto:*

- *Asegúrese que ninguna caja de diálogo aparezca en el equipo servidor (excepto para una necesidad específica). Para hacer esto, es necesario llamar al comando con el parámetro * o >.*
- *En caso de un problema con la impresora (sin papel, impresora desconectada, etc.), no se genera mensaje de error.*

Ejemplo

*El siguiente ejemplo selecciona todos los registros en la tabla [Personas]. El comando **DISPLAY SELECTION** es entonces llamado para mostrar los registros y permitir al usuario seleccionar los registros a imprimir. Finalmente, utiliza los registros seleccionados con el comando **USE SET**, y los imprime con **PRINT SELECTION**:*

```
ALL RECORDS([Personas]) ` Selección de todos los registros
DISPLAY SELECTION([Personas];*) ` Visualización de los registros
USE SET("UserSet") ` Utilizar únicamente los registros seleccionados por el usuario
PRINT SELECTION([Personas]) ` Imprimir los registros seleccionados por el usuario
```

PRINT SETTINGS

PRINT SETTINGS {(dialType)}

Parámetro	Tipo		Descripción
dialType	Entero largo	➡	Cajas de diálogo a mostrar

Descripción

*El comando **PRINT SETTINGS** provoca la visualización de uno o dos cajas de diálogo de parámetros de impresión. Este comando debe llamarse antes de una serie de comandos **Print form** o el comando **OPEN PRINTING JOB**.*

*El parámetro opcional tipoDial permite configurar la visualización de las cajas de diálogo de impresión. Puede utilizar una de las siguientes constantes del tema **Opciones de impresión**. Las cajas de diálogo de impresión que aparecen dependerán de la versión de 4D, como se muestra en la siguiente tabla:*

typeDial (constante)	4D 64 bits	4D 32 bits
0 o se omite	Impresión	Formato de impresión+Impresión
1 (Page setup dialog)	Formato de impresión	Formato de impresión
2 (Print dialog)	Impresión (= 0 o se omite)	Impresión

***Nota:** la caja de diálogo de impresión contiene la opción **Preview on Screen** que permite al usuario previsualizar su trabajo de impresión. Puede preseleccionar o deseleccionar esta opción llamando **SET PRINT PREVIEW** antes de llamar **PRINT SETTINGS**.*

Ejemplo

*Ver ejemplo para el comando **Print form**.*

Variables y conjuntos del sistema

Si el usuario hace clic en OK en ambas cajas de diálogo, la variable sistema OK toma el valor 1. De lo contrario, la variable sistema OK toma el valor 0.

Print settings to BLOB

Print settings to BLOB (confImp) -> Resultado

Parámetro	Tipo		Descripción
confImp	BLOB	⇐	Configuración de impresión actual
Resultado	Entero largo	⇒	Código de estado: 1=Operación exitosa, 0=Sin impresora actual

Descripción

El comando **Print settings to BLOB** guarda los ajustes de impresión 4D actuales en el BLOB confImp. El parámetro confImp almacena todos los valores utilizados para la impresión:

- Parámetros de diseño tales como papel, orientación, escala...
- Parámetros de impresión tales como el número de copias, fuente del papel...

Este comando debe ser usado en conjunto con el comando **BLOB to print settings**. Estos comandos le permiten guardar la configuración de impresión actual del usuario y recargarla después para que los usuarios no tengan que especificar sus parámetros cada vez que inician un trabajo de impresión. Además, permite mantener la configuración de la impresora "privada" (específica para el controlador de la impresora) no disponible como parámetros de impresión estándar.

El BLOB generado no debe ser modificado por programación; sólo puede ser utilizado por el comando **BLOB to print settings**.

El comando devuelve 1 si el BLOB se ha generado correctamente y 0 si no se selecciona ninguna impresora actual.

Windows / OS X

El BLOB confImp se puede guardar y leer en ambas plataformas. Sin embargo, incluso si algunos ajustes de impresión son comunes, algunos otros son específicos de la plataforma y dependen de los controladores y las versiones de sistema. Si el mismo BLOB confImp se comparte entre ambas plataformas, es posible que pierda partes de información.
Cuando se utiliza en un entorno heterogéneos, con el fin de restaurar la máxima configuración disponible para cada plataforma (y no sólo la parte común), se recomienda que maneje dos BLOBs confImp, uno para cada plataforma.

Ejemplo

Usted desea almacenar la configuración de impresión actual en el disco:

```
C_BLOB(curSettings)
PRINT SETTINGS  //muestra el diálogo de configuración de impresión al usuario
If(OK=1)
    $err:=Print settings to BLOB(curSettings)
    If($err=1)
        BLOB TO DOCUMENT(Get 4D folder+"current4Dsettings.blob";curSettings)
    Else
        ALERT("No hay ninguna impresora seleccionada")
    End if
End if
```

PRINTERS LIST

PRINTERS LIST (arrayNoms {; arrayNomsAlt {; arrayModelos}})

Parámetro	Tipo		Descripción
arrayNoms	Array texto	⇐	Nombres de las impresoras
arrayNomsAlt	Array texto	⇐	Windows: Ubicación de las impresoras Mac OS: Nombres personalizados de las impresoras
arrayModelos	Array texto	⇐	Modelos de impresoras

Descripción

*El comando **PRINTERS LIST** llena el (los) array(s) pasados como parámetro(s) con los nombres y opcionalmente con la ubicación o nombres personalizados y los modelos de impresión disponibles para el equipo.*

***Nota:** si las impresoras se manejan utilizando un servidor de impresión (spooler), se devuelve la ruta de acceso completa (bajo Windows) o el nombre del spooler (bajo Mac OS).*

Pase en el parámetro arrayNoms el nombre de un array de texto. Después de la ejecución del comando, este array contendrá los nombres de las impresoras disponibles. Bajo Mac OS, este será el "sistema" fijo de nombres.

Puede pasar un segundo array opcional, arrayNomsAlt. El contenido de este array dependerá de la plataforma:

- *Bajo Windows, para cada impresora, usted obtiene su ubicación en red (o puerto local).*
- *Bajo Mac OS, para cada impresora, usted obtiene su nombre personalizado, el cual puede se modificado por el usuario. Este nombre puede utilizarse, por ejemplo, en cajas de diálogos.*

*El parámetro opcional arrayModelos permite recuperar el modelo de cada impresora. (**Nota**: este parámetro no es soportado en las versiones Mac 32 bits de 4D).*

*Utilice los comandos **SET CURRENT PRINTER** y **Get current printer** para modificar u obtener la impresora seleccionada en 4D. Debe pasar los nombres devueltos en el primer array (arrayNoms).*

Bajo Windows, el nombre de una impresora puede ser modificado manualmente al nivel del sistema de operación. Por otra parte, su ubicación y su modelo están asociados a sus características físicas. Por lo tanto, usted puede utilizar los valores de array opcionales para verificar las características de la impresora seleccionada, generalmente, usted puede verificar que todos los equipos de los clientes utilizan la misma impresora.

Bajo Mac OS, esta verificación puede llevarse a cabo utilizando el nombre de la impresora (nombre del servidor de impresión), que es el mismo para cada equipo que está conectado.

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si el comando ha sido ejecutado correctamente; de lo contrario, toma el valor 0 y los arrays se devuelven vacíos.

Printing page

Printing page -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Número de la página que se está imprimiendo

Descripción

Printing page *devuelve el número de la página en impresión. Puede utilizarse sólo cuando esté imprimiendo con [#cmd id="60"/] o con el menú Impresión en el entorno Diseño.*

Ejemplo

El siguiente ejemplo cambia la posición de los números de página en un informe de manera que el informe pueda ser reproducido en un formato de dos lados. El formulario par el informe tiene dos variables que muestran los números de página. Una variable en la esquina inferior izquierda (vNumPagIzq) imprimirá los números de páginas pares. Una variable en la esquina inferior derecha (vNumPagDer) imprimirá los números de las páginas impares. El ejemplo prueba si el número de la página es par o impar, luego borra y utiliza las variables apropiadas:

```
Case of
   :(Form event=On Printing Footer)
      If((Printing page% 2)=0) ` Modulo es 0 para un número de página par
         vNumPagIzq:=String(Printing page) ` Fijar el número de página a la izquierda
         vNumPagDer:="" ` Borrar el número de página a la derecha
      Else ` De lo contrario, el número de página es impar
         vNumPagIzq:="" ` Borrar el número de página a la izquierda
         vNumPagDer:=String(Printing page) ` Fijar el número de página a la derecha
      End if
End case
```

SET CURRENT PRINTER

SET CURRENT PRINTER (nomImpr)

Parámetro	Tipo		Descripción
nomImpr	Cadena	➡	Nombre de la impresora a utilizar

Descripción

El comando ***SET CURRENT PRINTER*** *designa la impresora a utilizar para imprimir con la aplicación 4D actual.*

Pase el nombre de la impresora a seleccionar en el parámetro nomImpr. Para obtener una lista de impresoras disponibles, utilice de antemano el comando ***PRINTERS LIST****.*
Si pasa una cadena vacía en nomImpr, se utilizará la impresora actual definida en el sistema.

SET CURRENT PRINTER *le permite designar la impresora PDF genérica del sistema con el fin de imprimir archivos PDF. El valor a utilizar depende de la versión del sistema operativo, así como de la de 4D.*

- ***Windows 8 y versiones anteriores:***
  4D se basa en el driver PDFCreator para facilitar la impresión de documentos PDF con Windows (ver la sección ***Integración del driver PDFCreator bajo Windows****). Para imprimir un documento PDF, en el parámetro nomImp, ase el nombre de la impresora virtual que fue instalada por el driver PDFCreator. Por defecto, el nombre de la impresora virtual es "PDFCreator". Sin embargo, este nombre puede haber sido modificado cuando se instaló el driver. Con el fin de que 4D busque automáticamente y utilice el nombre de la impresora virtual, incluso si se ha personalizado, en el parámetro nomImpr debe pasar la siguiente constante (tema* ***Opciones de impresión****):*

Constante	Tipo	Valor	Comentario
PDFCreator Printer name	Cadena	PDFCreator	Visualización de la caja de impresión

- ***A partir de Windows 10:***
  Windows 10 incluye un driver de impresión PDF nativo, que permite a 4D crear directamente los PDFs sin que sea necesario utilizar un driver de terceros tal como PDFCreator.
  TEl nombre del driver es "Microsoft Print to PDF" (ver el ejemplo que se encuentra en la sección ***Integración del driver PDFCreator bajo Windows****).*

- ***En OS X y a partir de Windows 10 (4D v15 R5 64 bits o superior):***
  Una constante que se encuentra en el tema ***Opciones de impresión*** *le permite designar la impresora PDF genérica de forma automática, independientemente de la plataforma. Esto facilita la escritura de código genérico.*

Constante	Tipo	Valor	Comentario
Generic PDF driver	Cadena	_4d_pdf_printer	***Nota:*** *esta funcionalidad no está disponible en las versiones 32 bits de 4D.* ◦ *En OS X, declara el driver predeterminado como impresora actual. Este driver no es visible y no está en la lista devuelta por el comando* ***PRINTERS LIST****. la ruta de acceso al documento PDF se debe definir utilizando el comando* ***SET PRINT OPTION****, si no, se devuelve el error 3107.* ◦ *En Windows, declara el driver PDF de Windows (llamado "Microsoft Print to PDF") como impresora actual. Esta constante está disponible en Windows 10 únicamente, cuando está instalada la opción PDF. Con otras versiones de Windows, o cuando no hay ningún driver PDF disponible, el comando no hace nada y la variable OK toma el valor 0.*

El comando ***SET CURRENT PRINTER*** *debe llamarse antes de* ***SET PRINT OPTION****, de manera que las opciones disponibles correspondan a la impresora seleccionada. Por otra parte,* ***SET CURRENT PRINTER*** *debe llamarse después de* ***PAGE SETUP****, de lo contrario los parámetros de la impresora se pierden.*

Este comando puede utilizarse con los comandos ***PRINT SELECTION****,* ***PRINT RECORD****,* ***Print form*** *y* ***QR REPORT****, y se aplica a todas las impresiones de 4D, incluyendo en el modo Diseño.*
Los comandos de impresión deben llamarse obligatoriamente con el parámetro > (donde sea pertinente) de manera que los parámetros especificados no se pierdan.

Variables y conjuntos del sistema

Si la selección de impresora se lleva a cabo correctamente, la variable sistema OK toma el valor 1. Si ocurre lo contrario (por ejemplo si no se encuentra la impresora designada), la variable sistema OK toma el valor 0 y la impresora actual permanece sin cambios.

Ejemplo

Creación de un documento PDF bajo Windows 10 con 4D Developer Edition 64 bits:

```
C_TEXT($pdfpath)
$pdfpath:=System folder(Desktop)+"test.pdf"
SET CURRENT PRINTER(Generic PDF driver)
SET PRINT OPTION(Destination option;2;$pdfpath)
ALL RECORDS([Table_1])
```

```
PRINT SELECTION([Table_1];*)
SET CURRENT PRINTER("")
```

SET PRINT MARKER

SET PRINT MARKER (markNum ; posicion {; *})

Parámetro	Tipo		Descripción
markNum	Entero largo	➡	Número de marcador
posicion	Entero largo	➡	Nueva posición del marcador
*	Operador	➡	Si se pasa = mover los marcadores siguientes Si se omite = no mover los marcadores siguientes

Descripción

*El comando **SET PRINT MARKER** permite definir la posición de un marcador durante la impresión. Combinado con los comandos **Get print marker**, **OBJECT MOVE** o **Print form**, este comando le permite ajustar el tamaño de las áreas de impresión.*

***SET PRINT MARKER** puede utilizarse en dos contextos:*

- *durante el evento de formulario On header, en el contexto de los comandos **PRINT SELECTION** y **PRINT RECORD**.*
- *durante el evento de formulario On Printing Detail, en el contexto del comando **Print form**. Esta operación facilita la impresión de informes personalizados (ver ejemplo).*
  El efecto del comando está limitado a la impresión; ninguna modificación aparece en la pantalla. Las modificaciones realizadas a los formularios no se guardan.

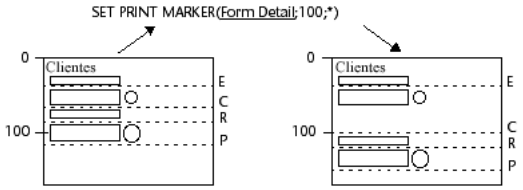
Pase en el parámetro markNum una de las constantes del tema :

Constante	Tipo	Valor
Form break0	Entero largo	300
Form break1	Entero largo	301
Form break2	Entero largo	302
Form break3	Entero largo	303
Form break4	Entero largo	304
Form break5	Entero largo	305
Form break6	Entero largo	306
Form break7	Entero largo	307
Form break8	Entero largo	308
Form break9	Entero largo	309
Form detail	Entero largo	0
Form footer	Entero largo	100
Form header	Entero largo	200
Form header1	Entero largo	201
Form header10	Entero largo	210
Form header2	Entero largo	202
Form header3	Entero largo	203
Form header4	Entero largo	204
Form header5	Entero largo	205
Form header6	Entero largo	206
Form header7	Entero largo	207
Form header8	Entero largo	208
Form header9	Entero largo	209

En posicion, pase la nueva posición deseada, expresada en píxeles.

*Si pasa el parámetro opcional *, todos los marcadores ubicados debajo del marcador especificado por markNum se moverán el mismo número de píxeles y en la misma dirección que este marcador cuando se ejecuta el comando. **Advertencia:** en este caso, los objetos presentes en las áreas situadas debajo del marcador también se mueven.*

*Cuando el parámetro * se utiliza, es posible posicionar el marcador markNum más allá de la posición inicial de los marcadores que le siguen, estos últimos marcadores se moverán simultáneamente.*



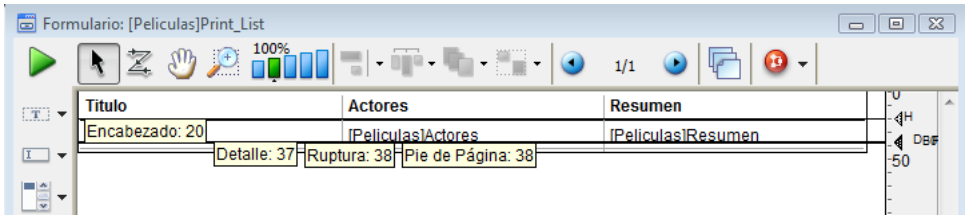
Notas:

• Este comando modifica sólo la posición de los marcadores existentes. No permite la adición de marcadores. Si designa un marcador que no existe en el formulario, el comando no hará nada.
*• El funcionamiento de los marcadores de impresión en modo Diseño se conserva: un marcador no puede ir más arriba del que lo precede, ni más abajo del que lo sigue (cuando el parámetro * no se utiliza).*

Ejemplo

Este ejemplo completo permite generar la impresión de un informe de tres columnas, la altura de cada línea se calcula de acuerdo a los contenidos de los campos.

El formulario de salida utilizado para la impresión es el siguiente:



*El evento de formulario On Printing Detail fue seleccionado para el formulario (recuerde que sin importar en que área se imprima, el comando **Print form** sólo genera este tipo de evento de formulario).*
Para cada registro, la altura de la línea debe estar adaptada de acuerdo a los contenidos de la columna "Actores" o "Resumen" (columna tiene la mayoría del contenido). Este es el resultado deseado:

Titulo	Actores	Resumen
The Avengers	Ralph Fiennes, Uma Thurman, Sean Connery, Jim Broadbent, Patrick Macnee, Fiona Shaw, Eddie Izzard, Eileen Atkins	"The Avengers", the popular TV series from the sixties, is brought to the big screen with a mix of humour, retro fashion and action. Ralph Fiennes plays the role of well dressed John Steed and Uma Thurman is the beautiful Emma Peel dressed in a jumpsuit. Our two special agents fight crime in style. Sean Connery plays Sir August De Wynter, an evil genious that wants to take over the world with his high-tech weather machine. Unleashing snow storms or heatwaves, and armed with remote-controlled killer bees, this nut is a real menance to society. But all these climate changes won't stop our two heroes. A cup of tea, anyone?
20,000 Leagues Under The Sea		"The year is 1866, when New England's fishing harbors are the scene for a creature of unknown origin destroying ships at sea. It is the job of Professor Pierre Aronnax, a marine expert, and Ned Land, the iron-willed sailor, to learn the truth of the monster roaming the seas. The great novelist, Jules Verne, described this perilous journey to the darkest depths of the sea with Capcain Nemo aboard the Nautilus."
The Adventures Of Ichabod And Mr. Toad (Walt Disney C	Bing Crosby, Basil Rathbone, Eric Blore, Pat O'Malley, John McLeish, Colin Campbell, Campbell Grant, Claud Allister	Hang on for the wild motorcar ride of J. Thaddeus Toad as he drives his friends Mole, Rat and Angus MacBadger into a worried frenzy! Then meet the spindly Ichabod Crane, who dreams of sweeping beautiful Katrina Van Tassel off her feet, despite opposition from town bully Brom Bones, who also has his eye on Katrina. The comic rivalry introduces Ichabod to the legend of the Headless Horseman resulting in a heart-thumping climax. Wonderfully narrated by Basil Rathbone and Bing Crosby, The Adventures Of Ichabod And Mr. Toad brims with high-spirited adventure, brilliant animation and captivating music you'll want to share with your family time and again.
Mission To Mars	Gary Sinise, Tim Robbins, Don Cheadle, Jerry O'Connell, Connie Nielsen	From the director of Mission Impossible comes the thrilling, eye-popping science-fiction adventure Mission To Mars - starring Gary Sinise (Snake Eyes) and Tim Robbins (Austin Powers: The Spy Who Shagged Me). The year is 2020, and the first manned mission to Mars, commanded by Luke Graham (Don Cheadle - Out Of Sight), lands safely on the red planet. But the Martian landscape harbors a bizarre and shocking secret that leads to a mysterious disaster so catastrophic, it decimates the crew. Haunted by a cryptic last message from Graham, NASA launches the Mars Recovery Mission to investigate and bring back survivors - if there are any. Confronted with nearly insurmountable dangers, but propelled by deep friendship, the team finally lands on Mars and makes a discovery so amazing, it takes your breath away. Mission To Mars is an action-packed rocket ride that will enthrall you with its stunning special effects and keep you on the edge of your seat.
The Abyss: Special Edition	Ed Harris, Mary Elizabeth Mastrantonio, Michael Biehn, Leo Burmeister, Todd Graff, John Bedford Lloyd, Kimberly Scott	In this thrilling, underwater action-adventure from writer-director James Cameron (Titanic, Terminator 2: Judgment Day, Aliens), a civilian oil-rig crew is recruited to conduct a search-and-rescue effort when a nuclear submarine mysteriously sinks. One diver (Ed Harris) soon finds himself on a spectacular odyssey over 25,000 feet below the ocean's surface, where he confronts a mysterious force that has the power to change the world or destroy it. Includes both the Special Edition, with 28 minutes of additional footage, and the original theatrical version, along with the 80-minute documentary Under Pressure: Making The Abyss, and much more.
Absence Of The Good	Stephen Baldwin, Rob Knepper, Shawn Hutt, Allen Garfield, Silas Weir Mitchell, Tyne Daly	One cop. One killer. No clues. No time. After his son is shot to death at school, Detective Caleb Barnes (Stephen Baldwin) loses touch with his soul. When a series of seemingly unrelated murders plagues Salt Lake City, the detective hides his grief in search for the killer. Hampered by a lack of clues and his commander's unrelenting pressure, Caleb painstakingly unravels a tangled web, exposing a malignant family history of abuse and murder.

El método de proyecto de impresión es el siguiente:

```
C_LONGINT(vLaltura_imp;$vLaltura;vLaltura_impreso)
C_STRING(31;vSprint_area)
PAGE SETUP([Peliculas];"List_Imp3")
GET PRINTABLE AREA(vLaltura_imp)
vLaltura_impreso:=0
ALL RECORDS([Peliculas])

vSprint_area:="Encabezado" `Impresión del área de encabezado
$vLaltura:=Print form([Peliculas];"List_Imp3";Form header)
$vLaltura:=21 `Altura fija
vLaltura_impreso:=vLaltura_impreso+$vLaltura

While(Not(End selection([Peliculas])))
   vSprint_area:="Detalle" `Impresión del área de detalle
   $vLaltura:=Print form([Peliculas];"List_Imp3";Form detail)
 `El cálculo del detalle se lleva a cabo en el método de formulario
   vLaltura_impreso:=vLaltura_impreso+$vLaltura
   If(OK=0) `CANCEL ha sido ejecutado en el método de formulario
      PAGE BREAK
      vLaltura_impreso:=0
```

```
        vSprint_area:="Encabezado" `Reimpresión del área de encabezado
        $vLaltura:=Print form([Peliculas];"List_Imp3";Form header)
        $vLaltura:=21
        vLaltura_impreso:=vLaltura_impreso+$vLaltura
        vSprint_area:="Detalle"
        $vLaltura:=Print form([Peliculas];"List_Imp3";Form detail)
        vLaltura_impreso:=vLaltura_impreso+$vLaltura
    End if
    NEXT RECORD([Peliculas])
End while
PAGE BREAK `Asegúrese de que la última página se imprima
```

El método de formulario List_Imp3 es el siguiente:

```
C_LONGINT($l;$t;$r;$b;$fixed_wdth;$exact_hght;$l1;$t1;$r1;$b1)
C_LONGINT($final_pos;$i)
C_LONGINT($detalle_pos;$encabezado_pos;$altura_a_imprimir;$altura_restante)

Case of
    :(vSprint_area="Detalle") `Impresión del detalle en proceso
        OBJECT GET COORDINATES([Peliculas]Actores;$l;$t;$r;$b)
        $largo_fijo:=$r-$l `Cálculo del tamaño del campo tipo texto Actores
        $altura_exact:=$b-$t
        OBJECT GET BEST SIZE([Peliculas]Actores;$largo;$alto;$largo_fijo)
    `Tamaño óptimo del campo de acuerdo a su contenido
        $movimiento:=$alto-$altura_exact

        OBJECT GET COORDINATES([Peliculas]Resumen;$l1;$t1;$r1;$b1)
        $largo_fijo:=$r1-$l1 `Cálculo del tamaño del campo tipo texto Resumen
        $altura_exact1:=$b1-$t1
        OBJECT GET BEST SIZE([Peliculas]Resumen;$largo1;$alto1;$largo_fijo)
    `Tamaño óptimo del campo de acuerdo a su contenido
        $movimiento1:=$alto1-$altura_exact1
        If($movimiento1>$movimiento)
    `Determinamos el campo más alto
            $movimiento:=$movimiento1
        End if

        If($movement>0)
            $posicion:=Get print marker(Form detail)
            $final_pos:=$posicion+$movimiento
    `Nos movemos al marcador Detalle y a los que siguen
            SET PRINT MARKER(Form detail;$final_pos;*)
    `Redimensionamiento de las áreas de texto
            OBJECT MOVE([Peliculas]Actores;$l;$t;$r;$hght+$t;*)
            OBJECT MOVE([Peliculas]Resumen;$l1;$t1;$r1;$alto1+$t1;*)

    `Redimensionamiento de las líneas de división
            OBJECT GET COORDINATES(*;"H1Linea";$l;$t;$r;$b)
            OBJECT MOVE(*;"H1Line";$l;$final_pos-1;$r;$final_pos;*)
            For($i;1;4;1)
                GET OBJECT RECT(*;"VLinea"+String($i);$l;$t;$r;$b)
                OBJECT MOVE(*;"VLinea"+String($i);$l;$t;$r;$final_pos;*)
            End for
        End if

    `Cálculo del espacio disponible
        $detalle_pos:=Get print marker(Form detail)
        $encabezado_pos:=Get print marker(Form header)
        $altura_a_imprimir:=$detalle_pos-$encabezado_pos
        $altura_restante:=altura_impreso-vLaltura_impreso
        If($altura_restante<$altura_a_imprimir) `Altura insuficiente
            CANCEL `Pasar la línea a la siguiente página
        End if
End case
```

SET PRINT OPTION

SET PRINT OPTION (opcion ; valor1 {; valor2})

Parámetro	Tipo		Descripción
opcion	Entero largo	➡	Número de opción o código de opción PDF
valor1	Entero largo, Texto	➡	Valor 1 de la opción
valor2	Entero largo, Texto	➡	Valor 2 de la opción

Descripción

El comando SET PRINT OPTION se utiliza para modificar por programación el valor de una opción de impresión.
*Cada opción definida utilizando este comando se aplica a toda la base y durante toda la sesión siempre que no se llame otro comando que modifique los parámetros de impresión (**PRINT SETTINGS**, **PRINT SELECTION** sin el parámetro > parámetro, etc.). Si se ha abierto un trabajo de impresión, la opción está configurada para el trabajo y no puede modificarse mientras el trabajo no haya terminado.*

*El parámetro opcion le permite indicar la opción a modificar. Puede pasar una de las constantes predefinidas del tema **Opciones de impresión**, o un código de opción PDF (utilizable con el driver PDFCreator bajo Windows únicamente).*
Pase en los parámetros valor1 y valor2 (opcionalmente) los nuevos valores de la opcion especificada. El número y naturaleza de los valores pasados depende del tipo de opción especificada.

Utilizar un número de opción (constante)

La siguiente tabla lista las opciones y sus posibles valores:

Constante	Tipo	Valor	Comentario
Paper option	Entero largo	1	Si sólo utiliza valor1, que contiene el nombre del papel. Si se utilizan los dos parámetros, valor1 contiene el ancho del papel y valor2 contiene el alto del papel. El ancho y el alto se expresan en píxeles de pantalla. Utilice el comando **PRINT OPTION VALUES** para obtener el nombre, el alto y el ancho de todos los formatos de papel que ofrece la impresora.
Orientation option	Entero largo	2	valor1 únicamente: 1=Retrato, 2=Paisaje. Si se utiliza una opción de orientación diferente, **GET PRINT OPTION**devuelve 0 en valor1. **Versiones 64 bits**: esta opción se puede llamar desde una tarea de impresión, lo que significa que puede cambiar de vertical a horizontal, o viceversa, durante el mismo trabajo de impresión.
Scale option	Entero largo	3	valor1 únicamente: valor de la escala en porcentaje. Tenga cuidado, algunas impresoras no permiten modificar la escala. Si pasa un valor no válido, la propiedad se reinicia al 100% en el momento de la impresión.
Number of copies option	Entero largo	4	valor1 únicamente: número de copias a imprimir.
Paper source option	Entero largo	5	(Windows únicamente) valor1 únicamente: número correspondiente al índice, en el array de bandejas devuelto por el comando **PRINT OPTION VALUES**, de la bandeja de papel a utilizar. Esta opción sólo se puede utilizar en Windows.
Color option	Entero largo	8	(Windows únicamente) valor1 únicamente: código que especifica el modo para el manejo del color: 1=Blanco y negro (monocromo), 2=Color. **Versiones 64 bits**: esta opción no está disponible en versiones 4D de 64 bits (obsoleto)
Destination option	Entero largo	9	valor1: código que especifica el tipo de destino de la impresión: 1=Impresora, 2=Archivo (PC)/PS (Mac), 3=Archivo PDF, 5=Pantalla (opción del driver OS X). Si valor1 es diferente de 1 o 5, valor2 contiene un nombre de ruta para el documento resultante. Esta ruta se utilizará hasta que se especifique otra ruta. Si un archivo con el mismo nombre ya existe en el lugar de destino, será sustituido. Con **GET PRINT OPTION**, si el valor actual no está en la lista predefinida, valor1 contiene -1 y la variable sistema OK toma el valor 1. Si ocurre un error, valor1 y la variable sistema OK toman el valor 0. **Nota**: en Windows, puede definir el destino de impresión 3 (archivo PDF) cuando el driver PDF Creator ha sido instalado. Cuando se pasan los valores (9;3;ruta), 4D inicia automáticamente una impresión PDF "silenciosa" que tiene en cuenta los códigos de opción pasados (note que si pasa una cadena vacía en valor2 u omite este parámetro, aparece el diálogo de guardar archivo en el momento de la impresión.) Después de la impresión, los parámetros actuales se restauran. Esto simplifica el control de las impresiones PDF por 4D y permite escribir código multiplataforma. Cuando los valores (9;3;ruta) no se transmiten, la impresión no es controlada por 4D y los eventuales códigos de opciones de PDF Creator se ignoran.
Double sided option	Entero largo	11	(Windows únicamente) valor1: 0=Un solo lado o estándar, 1=Doble cara. Si valor1=1, valor2 contiene la unión: 0=Izquierda (valor predeterminado), 1=Unión superior. **Nota**: esta opción sólo se puede utilizar en Windows.
Spooler document name option	Entero largo	12	valor1 únicamente: nombre del documento de impresión actual, que aparece en la lista de documentos de la cola de impresión. El nombre definido para esta instrucción se utilizará para todos los documentos de impresión de la sesión hasta que un nuevo nombre o una cadena vacía no se pase. Para utilizar o restablecer el funcionamiento normal (usando el nombre del método en el caso de un método, el nombre de la tabla para un registro, etc.), pase una cadena vacía en valor1.
Mac spool file format option	Entero largo	13	(Mac únicamente) valor1 únicamente: 0=impresión en modo PDF (valor por defecto) 1 = impresión en modo PostScript. **Notas:** - Esta opción no tiene efecto en Windows.. - En OS X, la impresión se realiza en formato PDF de forma predeterminada. Sin embargo, el driver de impresión PDF no es compatible con imágenes PICT con información PostScript encapsulada, estas imágenes se generan, particularmente, por los softwares de dibujo vectorial. Para evitar este problema, esta opción le permite modificar el modo de impresión a utilizar en OS X para la sesión actual. Tenga en cuenta la impresión en modo PostScript puede conducir a efectos secundarios no deseados. **Versiones 64 bits:** esta opción no es compatible; Es reemplazada por la opción Driver PDF générique del comando **SET CURRENT PRINTER**.
Hide printing progress option	Entero largo	14	(Mac únicamente) valor1 únicamente: 1=ocultar ventanas de progreso, 0= mostrar las ventanas de progreso de impresión (por defecto). Esta es una opción particularmente útil en el caso de impresión PDF en OS X. **Nota**: ya existe una opción de progreso de impresión accesible vía el cuadro de diálogo Propiedades de la base (página Interfaz). Sin embargo, se aplica globalmente a la aplicación y no oculta todas las ventanas bajo OS X.
Page range option	Entero largo	15	valor1=primera página a imprimir (valor por defecto 1) y (opcional) valor2=número de la última página a imprimir (valor por defecto -1 = fin del documento).
Legacy printing layer option	Entero largo	16	(Versiones 4D 64 bits para Windows únicamente) valor1 únicamente: 1=seleccionar la antigua capa de impresión GDI para todos los trabajos de impresión subsiguientes. 0=seleccionar la capa de impresión D2D (por defecto). **Versiones 64 bits:** este selector sólo es compatible con las aplicaciones 4D 64 bits mono usuario en Windows; se ignora en otras plataformas. Está destinado principalmente para permitir plug-ins de antigua generación imprimir dentro de tareas de impresión 4D.

Una vez fijado utilizando este comando, una opción de impresión se conservará durante toda la sesión para toda la aplicación 4D. Será utilizada por los comandos **PRINT SELECTION**, **PRINT RECORD**, **Print form** y **QR REPORT**, y para todas las impresiones 4D, incluyendo en modo Diseño.

Notas:

- *Es indispensable utilizar el parámetro opcional > con los comandos **PRINT SELECTION**, **PRINT RECORD** y **PAGE BREAK** para evitar reinicializar las opciones de impresión que fueron definidas utilizando el comando SET PRINT OPTION.*
- *El comando **SET PRINT OPTION** sólo opera con impresoras PostScript. Puede utilizar este comando con otros tipos de impresoras, tales como PCL o tinta, pero en este caso, es posible que algunas opciones no estén disponibles.*

Utilizar un código de opción PDF (Windows)

*Para poder utilizar un código de opción PDF en el parámetro opcion, debe haber instalado el driver PDFCreator en su entorno 4D (para mayor información, consulte la sección **Integración del driver PDFCreator bajo Windows**).*
Además, para que el código de opción sea tenido en cuenta, debe haber activado el driver de impresión PDF para 4D vía la siguiente instrucción:

```
SET PRINT OPTION(Destination option;3;nomArchivo)
```

De lo contrario, los códigos de opción se ignoran.

Un código de opción PDF es un valor de tipo texto que consta de dos partes, TipoOpcion y NombreOpción, combinados como "TipoOpcion:NombreOpcion". Esta es la descripción de este código:

- *TipoOpcion indica si usted especifica una opción nativa de PDFCreator o una opción de administración PDF de 4D. Se aceptan dos valores:*
  - ***PDFOptions** = opción nativa*
  - ***PDFInfo** = opción interna.*
- *NombreOpcion especifica la opción a definir (dependiendo del valor de TipoOpcion).*
  - *Si TipoOpcion = **PDFOptions**, puede pasar una de las numerosas opciones nativas de PDFCreator. Por ejemplo, la opción UseAutosave afecta el backup automático. Para poder modificar esta opción, pase "PDFOptions:UseAutosave" en el parámetro opcion y el valor a utilizar en el parámetro valor1. Para una descripción completa de las opciones nativas de PDFCreator, consulte la documentación del driver PDFCreator.*
  - *Si TipoOpcion = **PDFInfo**, puede pasar en NombreOpcion uno de los siguientes selectores específicos:*
    - ***Reset print**: permite reinicializar el esto de espera interna, particularmente, para salir del ciclo infinito. En este caso, valor1 no se utiliza.*
    - ***Reset standard options**: permite restablecer todas las opciones de PDFCreator a sus valores por defecto. Si hay una impresión en progreso, los parámetros por defecto se aplican luego de que termine la impresión. En este caso, valor1 no se utiliza.*
    - ***Start**: permite iniciar o detener el gestor de colas de impresión de PDFCreator. Pase 0 en valor1 para detenerlo y 1 para iniciarlo.*
    - ***Reset options**: permite reinicializar todas las opciones modificadas desde el inicio de la sesión utilizando el comando **SET PRINT OPTION** y el selector **PDFOptions**.*
    - ***Version**: permite leer el número de versión actual del driver PDF. Este selector sólo puede utilizarse con el comando **GET PRINT OPTION**. El número se devuelve en el parámetro valor1.*
    - ***Last error**: permite leer el último error devuelto por el driver PDFCreator. Este selector puede utilizarse únicamente con el comando **GET PRINT OPTION**. El número de error se devuelve en el parámetro valor1.*
    - ***Print in progress**: permite saber si 4D está esperando una impresión de PDFCreator. Este selector puede utilizarse únicamente con el comando **GET PRINT OPTION**. El parámetro valor1 devuelve 1 si 4D está esperando a PDFCreator y 0 de lo contrario.*
    - ***Job count**: permite conocer el número de trabajo en espera de la cola de impresión. Este selector puede utilizarse únicamente con el comando **GET PRINT OPTION**. El número de trabajos se devuelve en el parámetro valor1.*
    - ***Synchronous Mode**: permite definir el modo de sincronización entre las peticiones de impresión enviadas por 4D y el driver PDFCreator. Como 4D no puede obtener información relacionada con el estado actual de un trabajo de impresión que está en la cola de impresión, esta opción permite controlar la ejecución de tareas enviándolas únicamente cuando el esto del driver PDFCreator sea "libre". En este caso, 4D se sincroniza con el driver. Pase 0 en valor1 para que 4D envíe inmediatamente las peticiones de impresión (valor por defecto) y 1 para que 4D se sincronice y espere a que el driver haya terminado el trabajo antes de enviar otra tarea.*

***Nota:** luego de cada impresión, 4D restablece automáticamente los parámetros anteriores del driver PDFCreator con el fin de evitar toda interferencia con los otros programas que utilizan PDFCreator.*

Ejemplo 1

El siguiente método activa el driver PDF de manera que para imprimir todos los registros de la tabla en la ubicación C:\Test_PDF_X donde X es el número de secuencia del registro:

```
SET CURRENT PRINTER(PDFCreator Printer Name)
 // Bajo Windows, seleccione la impresora virtual instalada por PDFCreator
If(OK=1) // Si PDFCreator está instalado</p><p>ALL RECORDS([Table_1])
   For($i;1;Records in selection([Table_1]))
      SET PRINT OPTION(Destination option;3;"C:\\Test\\Test_PDF_"+String($i))
 // La opción de destino 3 lanza una tarea de impresión PDFCreator
      PRINT RECORD([Table_1];*)
      NEXT RECORD([Table_1])
   End for
 // Reinicialización de las opciones del driver PDFCreator
   SET PRINT OPTION("PDFInfo:Reset standard options";0)
End if
```

Ejemplo 2

En versiones 64 bits, el valor de Orientation option puede modificarse en el mismo trabajo de impresión (caso especial). Note que la opción debe haberse definido antes del comando ***PAGE BREAK****:*

```
ALL RECORDS([People])
PRINT SETTINGS
If(OK=1)
    OPEN PRINTING JOB
    SET PRINT OPTION(Orientation option;1) //portrait
    Print form([People];"Vertical_Form")

    SET PRINT OPTION(Orientation option;2) //landscape
    PAGE BREAK //must be called imperatively AFTER the option
    Print form([People];"Horiz_Form")
    CLOSE PRINTING JOB
End if
```

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si el comando ha sido ejecutado correctamente; de lo contrario, toma el valor 0.
Si pasa un código de opción invalido (opción no reconocida por PDFCreator por ejemplo), OK toma el valor 0.

Gestión de errores

Si el valor pasado por una opción es incorrecto o si no está disponible en la impresora, el comando devuelve un error (que puede interceptar utilizando un método de gestión de errores instalado por el comando ***ON ERR CALL****) y el valor actual de la opción permanece sin cambios.*

SET PRINT PREVIEW

SET PRINT PREVIEW (vista previa)

Parámetro	Tipo		Descripción
vista previa	Booleano	⇒	Previsualización en pantalla (TRUE), o Sin previsualización (FALSE)

Descripción

SET PRINT PREVIEW *le permite seleccionar o deseleccionar por programación la opción de previsualización en pantalla. Si pasa TRUE en vista previa, se selecciona Previsualización en pantalla, si pasa FALSE, se deseleccionará. Este parámetro es local para un proceso y no afecta la impresión de otros procesos o usuarios.*

Ejemplo

El siguiente ejemplo selecciona la opción Previsualización en pantalla para mostrar los resultados de una búsqueda en pantalla, y luego la deselecciona.

```
QUERY([Clientes])
If(OK=1)
   SET PRINT PREVIEW(True)
   PRINT SELECTION([Clientes];*)
   SET PRINT PREVIEW(False)
End if
```

SET PRINTABLE MARGIN

SET PRINTABLE MARGIN (izquierda ; superior ; derecha ; inferior)

Parámetro	Tipo		Descripción
izquierda	Entero largo	➡	Margen izquierda
superior	Entero largo	➡	Margen superior
derecha	Entero largo	➡	Margen derecha
inferior	Entero largo	➡	Margen inferior

Descripción

*El comando **SET PRINTABLE MARGIN** [#descv]permite asignar los valores de varias márgenes de impresión utilizando los comandos **Print form**, **PRINT SELECTION** y **PRINT RECORD**.*

Puede pasar uno de los siguientes valores en los parámetros izquierda, superior, derecha e inferior:

- *0 = utilizar las márgenes del papel*
- *-1 = utilizar las márgenes de la impresora*
- *valor > 0 = margen en píxeles (recuerde que 1 píxel en 72 dpi representa aproximadamente 0.4 mm)*

Los valores de los parámetros derecha e inferior son relativos a los bordes derecho e inferior del papel.

***Nota:** para mayor información sobre gestión de impresión y terminología en 4D, consulte la descripción del comando **GET PRINTABLE MARGIN**.*

*Por defecto, 4D basa sus impresiones en las márgenes de la impresora. Una vez se ejecuta el comando **SET PRINTABLE MARGIN**, los parámetros modificados se conservarán en el mismo proceso para toda la sesión.*

Ejemplo 1

El siguiente ejemplo le permite obtener el tamaño de la margen muerta:

```
SET PRINTABLE MARGIN(-1;-1;-1;-1)  `Define la marge de la impresora
GET PRINTABLE MARGIN($l;$t;$r;$b)
  `$l, $t, $r y $b corresponden a las márgenes muertas de la hoja
```

Ejemplo 2

El siguiente ejemplo le permite obtener el tamaño del papel:

```
SET PRINTABLE MARGIN(0;0;0;0)  `Define la margen del papel
GET PRINTABLE AREA($alto;$largo)
  `Para A4: $alto=842 ; $largo=595 píxeles
```

Subtotal

Subtotal (valores {; saltoPag}) -> Resultado

Parámetro	Tipo		Descripción
valores	Campo	➡	Campo o variable numérica donde quiere devolver el subtotal
saltoPag	Entero largo	➡	Nivel de ruptura para el cual efectuar un salto de página
Resultado	Real	⮌	Subtotal de valores

Descripción

Subtotal *devuelve el subtotal de valores para el nivel de ruptura actual o anterior.* ***Subtotal*** *sólo funciona cuando una selección ordenada se imprime con* ***PRINT SELECTION*** *o utilizando Imprimir en el entorno Diseño. El parámetro valores debe ser de tipo real, entero, o entero largo. Usted debe asignar el resultado de la función* ***Subtotal*** *a una variable ubicada en el área de ruptura del formulario.*

Advertencia: debe *ejecutar los comandos* ***BREAK LEVEL*** *y* ***ACCUMULATE*** *antes de cada informe de formulario para el cual quiera hacer proceso de ruptura y calcular subtotales. Ver la discusión al final de la descripción de este comando.*

El segundo parámetro (opcional) de la función ***Subtotal*** *se utiliza para provocar saltos de página durante la impresión. Si saltoPag es 0,* ***Subtotal*** *no genera ningún salto de página. Si saltoPag es igual a 1,* ***Subtotal*** *genera un salto de página para cada nivel de ruptura 1. Si saltoPag es igual a 2,* ***Subtotal*** *genera un salto de página para cada nivel de ruptura 1 y 2, etc.*

Consejo: *si ejecuta* ***Subtotal*** *desde un formulario de salida mostrado en pantalla, se generará un error, disparando un bucle infinito de actualizaciones entre el formulario y la ventana de error. Para salir de este bucle, presione Alt+Mayús (Windows) u Opción-Mayús (Macintosh) y haga clic en el botón Abortar en la ventana de error (probablemente tenga que hacer esto varias veces). Esto detiene temporalmente las actualizaciones de la ventana del formulario. Seleccione otro formulario de salida de manera que el error no se repita. Regrese al entorno Diseño y aísle la llamada a* ***Subtotal*** *para una prueba* ***Form event=On Printing Break*** *si tiene la intención de utilizar el mismo formulario de salida para la visualización y la impresión.*

Ejemplo

El siguiente ejemplo es un método de objeto en un área de ruptura de un formulario (B0, el área situada sobre el marcador B0). La variable vSalario está ubicada en el área de ruptura. La variable toma el valor del subtotal del campo Salario para este nivel de ruptura. El tratamiento de ruptura debe haber sido activado de antemano utilizando los comandos ***ACCUMULATE*** *y* ***BREAK LEVEL****.*

```
Case of
   :(Form event=On Printing Break)
      vSalario:=Subtotal([Empleados]Salario)
End case
```

Para mayor información sobre diseño de formulario con áreas de encabezado y de ruptura, consulte el manual de Diseño.

Activación de niveles de ruptura en los formularios de informes

Para poder generar informes con rupturas, debe activar el tratamiento de rupturas llamando los comandos ***BREAK LEVEL y ACCUMULATE****.*

Debe ejecutar ambos comandos antes de imprimir un informe de formulario. La función ***Subtotal*** *es necesaria para mostrar los valores en un formulario. Debe ordenar al menos el número de niveles de rupturas que necesite.*

Cuando utilice ***BREAK LEVEL y ACCUMULATE,*** *el proceso de imprimir un informe es generalmente como este:*

1. Seleccione los registros a imprimir.

2. Ordene los registros utilizando ***ORDER BY****. Ordene el mismo número de niveles como rupturas.*

3. Ejecute ***BREAK LEVEL y ACCUMULATE.***

4. Imprima el informe utilizando ***PRINT SELECTION****.*

La función ***Subtotal*** *es necesaria para mostrar valores en un formulario.*

Informes rápidos

- *QR BLOB TO REPORT*
- *QR Count columns*
- *QR DELETE COLUMN*
- *QR DELETE OFFSCREEN AREA*
- *QR EXECUTE COMMAND*
- *QR Find column*
- *QR Get area property*
- *QR GET BORDERS*
- *QR Get command status*
- *QR GET DESTINATION*
- *QR Get document property*
- *QR Get drop column*
- *QR GET HEADER AND FOOTER*
- *QR Get HTML template*
- *QR GET INFO COLUMN*
- *QR Get info row*
- *QR Get report kind*
- *QR Get report table*
- *QR GET SELECTION*
- *QR GET SORTS*
- *QR Get text property*
- *QR GET TOTALS DATA*
- *QR GET TOTALS SPACING*
- *QR INSERT COLUMN*
- *QR MOVE COLUMN*
- *QR NEW AREA*
- *QR New offscreen area*
- *QR ON COMMAND*
- *QR REPORT*
- *QR REPORT TO BLOB*
- *QR RUN*
- *QR SET AREA PROPERTY*
- *QR SET BORDERS*
- *QR SET DESTINATION*
- *QR SET DOCUMENT PROPERTY*
- *QR SET HEADER AND FOOTER*
- *QR SET HTML TEMPLATE*
- *QR SET INFO COLUMN*
- *QR SET INFO ROW*
- *QR SET REPORT KIND*
- *QR SET REPORT TABLE*
- *QR SET SELECTION*
- *QR SET SORTS*
- *QR SET TEXT PROPERTY*
- *QR SET TOTALS DATA*
- *QR SET TOTALS SPACING*

QR BLOB TO REPORT

QR BLOB TO REPORT (area ; BLOB)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
BLOB	BLOB	➡	BLOB que contiene el informe

Descripción

*El comando **QR BLOB TO REPORT** coloca el informe contenido en el blob en el área de informe rápido pasada en area.*
Si pasa un número de area incorrecto, se genera el error -9850.
Si el parámetro blob es incorrecto, se genera el error -9852.

Ejemplo 1

El siguiente código muestra en el área MiArea, un archivo de informe llamado "report.4qr" ubicado junto a la estructura de la base. El archivo de informe pudo haber sido creado con una versión anterior:

```
C_BLOB($doc)
C_LONGINT(MiArea)
DOCUMENT TO BLOB("report.4qr";$doc)
QR BLOB TO REPORT(MiArea;$doc)
```

Ejemplo 2

La siguiente instrucción recupera el informe rápido almacenado en Campo4 y lo muestra en MiArea:

```
QR BLOB TO REPORT(MiArea;[Tabla 1]Campo4)
```

QR Count columns

QR Count columns (area) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
Resultado	Entero largo	⮌	Número de columnas en el área

Descripción

*El comando **QR Count columns** devuelve el número de columnas presentes en el area del informe rápido.*
Si pasa un número de area inválido, se genera el error -9850.

Ejemplo

El siguiente código permite añadir una columna adicional a la derecha de la última columna del área:

```
$ColNb:=QR Count columns(MiArea)
QR INSERT COLUMN(MiArea;$ColNb+1;->[Tabla 1]Campo2)
```

QR DELETE COLUMN

QR DELETE COLUMN (area ; numColumna)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
numColumna	Entero largo	➡	Número de columna

Descripción

*El comando **QR DELETE COLUMN** borra del area la columna cuyo número se pasó en numColumna. Este comando no aplica a los informes de tablas cruzadas.*

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro numcolumna es incorrecto, se genera el error -9852.

Ejemplo

El siguiente ejemplo se asegura de que el informe sea listado y borra la tercera columna:

```
If(QR Get report kind(MyArea)=qr list report)
   QR DELETE COLUMN(MiArea;3)
End if
```

QR DELETE OFFSCREEN AREA

QR DELETE OFFSCREEN AREA (area)

Parámetro	***Tipo***		***Descripción***
area	*Entero largo*	➡	*Referencia del área a borrar*

Descripción

*El comando **QR DELETE OFFSCREEN AREA** borra de la memoria el área fuera de pantalla del informe rápido cuya referencia se pasa en el parámetro area.*

Si pasa un número de area incorrecto, se genera el error -9850.

QR EXECUTE COMMAND

QR EXECUTE COMMAND (area ; comando)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
comando	Entero largo	➡	Comando de menú a ejecutar

Nota de compatibilidad

*A partir de 4D v16, se incluye un nuevo editor **Informes rápidos (64 bits)** con las versiones 64 bits de 4D. La interfaz "ribbon" de este editor no es personalizable. Como resultado, la mayoría de los comandos de menú designados por las constantes siguientes no se pueden usar en el editor de informes rápidos de 64 bits. Las constantes compatibles se indican en la tabla.*

*Tenga en cuenta que es posible diseñar una interfaz totalmente personalizada utilizando el área incluida en el editor de 64 bits (ver **Crear un nuevo informe rápido**).*

Descripción

*El comando **QR EXECUTE COMMAND**ejecuta el comando de menú o el botón de la barra de herramientas cuya referencia se pasa en comando. Este comando se utiliza por lo general para ejecutar un comando de menú seleccionado por el usuario e interceptado en su código a través del comando **QR ON COMMAND**.*

*En comando, puede pasar una de las constantes del tema **QR Comandos**:*

Constante	Tipo	Valor	Comentario
qr cmd 4D View destination	Entero largo	2503	
qr cmd add column	Entero largo	2608	
qr cmd alt back color palette	Entero largo	1004	
qr cmd automatic width	Entero largo	2605	
qr cmd average	Entero largo	507	
qr cmd back color palette	Entero largo	1003	
qr cmd back colors toolbar	Entero largo	2052	
qr cmd bold	Entero largo	500	
qr cmd borders	Entero largo	2609	
qr cmd center justified	Entero largo	504	
qr cmd columns toolbar	Entero largo	2054	
qr cmd count	Entero largo	510	
qr cmd default justified	Entero largo	512	
qr cmd delete column	Entero largo	2601	
qr cmd disk file destination	Entero largo	2501	
qr cmd edit column	Entero largo	2603	
qr cmd font color palette	Entero largo	1002	
qr cmd font dropdown	Entero largo	1000	
qr cmd format	Entero largo	2606	
qr cmd generate	Entero largo	2008	Compatible editor 64 bits (uso del comando **QR RUN** recomendado)
qr cmd graph destination	Entero largo	2502	
qr cmd header and footer	Entero largo	2005	
qr cmd hide column	Entero largo	2602	
qr cmd hide line	Entero largo	2607	
qr cmd HTML file destination	Entero largo	2504	
qr cmd insert column	Entero largo	2600	
qr cmd italic	Entero largo	501	
qr cmd left justified	Entero largo	503	
qr cmd max	Entero largo	509	
qr cmd min	Entero largo	508	
qr cmd move left	Entero largo	3002	
qr cmd move right	Entero largo	3003	
qr cmd new	Entero largo	2000	
qr cmd open	Entero largo	2001	
qr cmd operators toolbar	Entero largo	2051	
qr cmd page setup	Entero largo	2006	Compatible editor 64 bits
qr cmd plain	Entero largo	511	
qr cmd presentation	Entero largo	2611	
qr cmd print preview	Entero largo	2007	Compatible editor 64 bits
qr cmd printer destination	Entero largo	2500	
qr cmd repeated values	Entero largo	2604	
qr cmd revert to save	Entero largo	2004	
qr cmd right justified	Entero largo	505	
qr cmd save	Entero largo	2002	
qr cmd save as	Entero largo	2003	
qr cmd standard deviation	Entero largo	513	
qr cmd standard toolbar	Entero largo	2053	
qr cmd style toolbar	Entero largo	2050	
qr cmd sum	Entero largo	506	
qr cmd totals spacing	Entero largo	2610	
qr cmd underline	Entero largo	502	

Si pasa un número de area inválido, se genera el error -9850.
Si pasa un número de comando incorrecto, se genera el error -9852.

QR Find column

QR Find column (area ; expresion) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
expresion	Cadena, Puntero	➡	Objeto de columna
Resultado	Entero largo	⮌	Número de columna

Descripción

*El comando **QR Find column** devuelve el número de la primera columna cuyo contenido corresponde a la expresion pasada en parámetro.*

expresion puede ser una cadena o un puntero.

***QR Find column** devuelve –1 si no se encuentra nada.*

Si pasa un número de area inválido, se genera el error -9850.

Ejemplo

El siguiente código permite recuperar el número de la columna que contiene el campo [G.NQR Tests]Trimestre y borra esa columna:

```
$NumColumn:=QR Find column(MiArea;->[G.NQR Tests]Trimestre)
```

o:
$NumColumn:=QR Find column (MiArea; "[G.NQR Tests]Trimestre")

```
If($NumColumn#-1)
   QR DELETE COLUMN(MiArea;$NumColumn)
End if
```

QR Get area property

QR Get area property (area ; propiedad) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
propiedad	Entero largo	➡	Elemento de interfaz
Resultado	Entero largo	⮌	1 = mostrado, 0 = oculto

Descripción

*El comando **QR Get area property** devuelve 0 si no se muestra el elemento de interfaz (barra de herramientas o barra de menús) pasado en propiedad; de lo contrario, devuelve 1.*

La barra de menús y la barra de herramientas son numeradas del 1 al 6 (de arriba a abajo) y el valor 7 se asocia al menú contextual.

Puede utilizar las constantes del tema para designar la interfaz del elemento:

Constante	Tipo	Valor	Comentario
qr view color toolbar	Entero largo	5	Muestra la barra de herramientas Colores (Mostrada=1, Oculta=0)
qr view column toolbar	Entero largo	6	Muestra la barra de herramientas Columnas (Mostrada=1, Oculta=0)
qr view contextual menus	Entero largo	7	Muestra los menús contextuales (Mostrados=1, Ocultos=0)
qr view menubar	Entero largo	1	Muestra la barra de menús (Mostrada=1, Oculta=0)
qr view operators toolbar	Entero largo	4	Muestra la barra de herramientas Operadores (Mostrada=1, Oculta=0)
qr view standard toolbar	Entero largo	2	Muestra la barra de herramientas estándar (Mostrada=1, Oculta=0)
qr view style toolbar	Entero largo	3	Muestra la barra de herramientas Estilo (Mostrada=1, Oculta=0)

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro propiedad es incorrecto, se genera el error -9852.

QR GET BORDERS

QR GET BORDERS (area ; columna ; linea ; borde ; grueso {; color})

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
columna	Entero largo	➡	Número de columna
linea	Entero largo	➡	Número de línea
borde	Entero largo	➡	Valor del borde
grueso	Entero largo	⬅	Grueso de línea
color	Entero largo	⬅	Color del borde

Descripción

*El comando **QR GET BORDERS** permite recuperar el estilo del borde de una celda determinada.*

area es la referencia del área del informe rápido.
columna es el número de columna de la celda.
linea designa el número de fila de la celda.

- *pase un valor entero positivo para designar el nivel (ruptura) subtotal correspondiente que está afectado.*
- *pase una de las siguientes constantes del tema **QR Filas para propiedades**:*

Constante	Tipo	Valor	Comentario
qr detail	Entero largo	-2	Área de detalle del informe
qr grand total	Entero largo	-3	Área total general
qr title	Entero largo	-1	Título del informe

*borde es el valor que indica qué borde de celda se afecta. Pase una de las constantes del tema **QR Bordes**:*

Constante	Tipo	Valor	Comentario
qr bottom border	Entero largo	8	Borde inferior
qr inside horizontal border	Entero largo	32	Borde interior horizontal
qr inside vertical border	Entero largo	16	Borde interior vertical
qr left border	Entero largo	1	Borde izquierdo
qr right border	Entero largo	4	Borde derecho
qr top border	Entero largo	2	Borde superior

***Nota:** a diferencia del comando **QR SET BORDERS**, **QR GET BORDERS** no acepta un valor acumulativo. Debe probar todos los parámetros por separado para tener una visión general del borde de celda.*

grueso es el grueso de la línea:

- *0 indica no linea*
- *1 indica un grueso de 1/4 de punto*
- *2 indica un grueso de 1/2 de punto*
- *3 indica un grueso de 1 punto*
- *4 indica un grueso de 2 puntos*

color is the color of the line; it returns the value of the color applied to the line segment.

color es el color de la línea; devuelve el valor del color aplicado a la línea.

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro columna es incorrecto, se genera el error -9852.
Si el parámetro linea es incorrecto, se genera el error -9853.
Si el parámetro borde es incorrecto, se genera el error -9854.
Si el parámetro area es incorrecto, se genera el error -9850.

QR Get command status

QR Get command status (area ; comando {; valor}) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
comando	Entero largo	➡	Número del comando
valor	Entero largo, Texto	⬅	Valor del subelemento seleccionado
Resultado	Entero largo	↩	Estado del comando

Descripción

El comando ***QR Get command status*** *devuelve 0 si el comando está inactivo ó 1 si está activo.*

valor devuelve el valor del subelemento seleccionado, si lo hay. Por ejemplo, si el comando que fue seleccionado es el menú ***Fuente*** *(1000) y la fuente seleccionada es "Arial", valor devuelve "Arial", o si el comando seleccionado es el menú de los color (1002, 1003 o 1004), valor devuelve el número del color.*

Puede utilizar este comando en dos tipos de contextos:

- *Como una instrucción simple para determinar si un comando está activo o no.*
- *En un método instalado por* ***QR ON COMMAND****, para permitirle conocer el subelemento seleccionado. En ese método, $1 es la referencia del área y $2 es el número del comando.*

En comando, puede pasar un valor o una de las constantes del tema .

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro comando es incorrecto, se genera el error -9852.

QR GET DESTINATION

QR GET DESTINATION (area ; tipo {; especificos})

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
tipo	Entero largo	⬅	Tipo de informe
especificos	Cadena, Variable	⬅	Específicos asociados al tipo de salida

Descripción

El comando ***QR GET DESTINATION*** *recupera el tipo de salida del informe para el área cuya referencia se pasó en area.*
Puede comparar el valor del parámetro tipo con las constantes del tema .
La siguiente tabla describe los valores que se pueden recuperar en los parámetros tipo y especificos:

Destino	Tipo (valor)	Específicos
Impresora	qr printer (1)	N.A.
Archivo texto	qr text file (2)	Ruta de acceso al archivo
4D View	qr 4D View area (3)	N.A.
4D Chart	qr 4D Chart area (4)	N.A.
Archivo HTML	qr HTML file (5)	Ruta de acceso al archivo HTML

Si pasa un número de area incorrecto, se genera el error -9850.

QR Get document property

QR Get document property (area ; propiedad) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
propiedad	Entero largo	➡	1 = Diálogo de impresión, 2 = Unidad del documento
Resultado	Entero largo	⮌	Valor de la propiedad

Descripción

El comando ***QR Get document property*** *permite recuperar el estado mostrado para la caja de diálogo de impresión o la unidad utilizada para el documento presente en area.*

En propiedad puede utilizar una de las constantes del tema :

Constante	Tipo	Valor	Comentario
qr printing dialog	Entero largo	1	Visualización de la caja de diálogo de impresión: • Si valor = 0, la caja de diálogo de impresión no se muestra antes de la impresión. • Si valor = 1, la caja de diálogo de impresión se muestra antes de la impresión (valor por defecto).
qr unit	Entero largo	2	Unidad del documento: • Si valor = 0, la unidad del documento es el punto. • Si valor = 1, la unidad del documento es el centímetro. • Si valor = 2, la unidad del documento es la pulgada.

- *Si propiedad es igual a 1, el comando aplica a la visualización de la caja de diálogo de impresión.*

  - Si valor es igual a 1, la caja de diálogo de impresión se muestra antes de la impresión.
  - Si valor es igual a 0, la caja de diálogo de impresión no se muestra antes de la impresión.
  El valor por defecto es 1.

- *Si propiedad es igual a 2, el comando aplica a la unidad del documento.*
  - Si valor es igual a 0, la unidad del documento es el punto.
  - Si valor es igual a 1, la unidad del documento es el centímetro.
  - Si valor es igual a 2, la unidad del documento es la pulgada.

Si pasa un número de area inválido, se genera el error -9850.
Si pasa un valor incorrecto del parámetro propiedad, se genera el error -9852.

QR Get drop column

QR Get drop column (area) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
Resultado	Entero largo	⮌	Espacio de soltar

Descripción

El comando ***QR Get drop column*** *devuelve un valor dependiendo de dónde se efectúe una acción soltar:*

- *si el valor es negativo, indica un número de columna (por ejemplo, -3 indica que una acción soltar se realizó en la columna número 3)*
- *si el valor es positivo, indica que la acción de soltar se realizó en un separador situado delante de la columna (por ejemplo, 3 indica que un soltar se efectúo en la columna 2). Recuerde que la acción de soltar no se tiene que lleva a cabo antes de una columna existente.*

Si pasa un número de area inválido, se genera el error -9850.

QR GET HEADER AND FOOTER

QR GET HEADER AND FOOTER (area ; selector ; tituloIzq ; tituloCent ; tituloDer ; alto {; imagen {; alinImag}})

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
selector	Entero largo	➡	1 = Encabezado, 2 = Pie de página
tituloIzq	Cadena	⬅	Texto mostrado a la izquierda
tituloCent	Cadena	⬅	Texto mostrado en el centro
tituloDer	Cadena	⬅	Texto mostrado a la derecha
alto	Entero largo	⬅	Altura del encabezado o pie de página
imagen	Imagen	⬅	Imagen a mostrar
alinImag	Entero largo	⬅	Atributo de alineación para la imagen

Descripción

*El comando **QR GET HEADER AND FOOTER** permite recuperar el contenido y el tamaño del encabezado o pie de página.*
selector le permite seleccionar el encabezado o el pie de página:

- *si selector es igual a 1, la información del encabezado se recuperará;*
- *si selector es igual a 2, la información del pie de página se recuperará.*

tituloIzq, tituloCent y tituloDer devuelve los valores para los encabezados/pies de páginas izquierdo, centro y derecha, respectivamente.
altura devuelve la altura del encabezado/pie de página, expresada en la unidad seleccionada para el informe.
imagen devuelve una imagen que se muestra en el encabezado o pie de página.
alinImag es el atributo de alineación para la imagen mostrada en el encabezado/pie de página.

- *Si alinImag devuelve 1, la imagen se alinea a la izquierda.*
- *Si alinImag devuelve 2, la imagen se centra.*
- *Si alinImag devuelve 3, la imagen se alinea a la derecha.*

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro selector es incorrecto, se genera el error -9852.

Ejemplo

El siguiente código recupera el contenido y la altura del título del encabezado y los muestra como alertas:

```
QR GET HEADER AND FOOTER(MiArea;1;$textIzq;$textCent;$texDer;$altura)
Case of
   :($textIzq#"")
      ALERT("El título a la izquierda es "+Char(34)+$textIzq+Char(34))
   :($textCent#"")
      ALERT("El título del centro es "+Char(34)+$textCent+Char(34))
   :($texDer#"")
      ALERT("El título de la derecha es "+Char(34)+$texDer+Char(34))
   Else
      ALERT("No hay título del encabezado en este informe.")
End case
ALERT("La altura del encabezado es "+String($altura))
```

QR Get HTML template

QR Get HTML template (area) -> Resultado

Parámetro	*Tipo*		*Descripción*
area	*Entero largo*	➡	*Referencia del área*
Resultado	*Texto*	⮌	*Código HTML utilizado como plantilla*

Descripción

*El comando **QR Get HTML template** devuelve la plantilla HTML utilizada actualmente por el área del informe rápido referenciada por area. El valor devuelto es de tipo texto e incluye la totalidad del código HTML utilizado como plantilla.*

Si no se definió una plantilla específica, se devuelve la plantilla por defecto. Por favor tenga en cuenta que no se devolverá ningún valor si el formato de destino HTML no fue definido para el informe, manualmente o por programación.

Si pasa un número de area inválido, se genera el error -9850.

QR GET INFO COLUMN

QR GET INFO COLUMN (area ; numColumna ; titulo ; objeto ; oculta ; tamaño ; valoresRepetidos ; formato {; varResultado})

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
numColumna	Entero largo	➡	Número de columna
titulo	Texto	⬅	Título de la columna
objeto	Texto	⬅	Objeto asignado a la columna
oculta	Entero largo	⬅	0 = visible, 1 = oculto
tamaño	Entero largo	⬅	Largo de la columna
valoresRepetidos	Entero largo	⬅	0 = no repetido, 1 = repetido
formato	Texto	⬅	Formato de visualización de los datos
varResultado	Texto	⬅	Nombre de la variable de fórmula

Descripción

Informes en lista

*El comando **QR GET INFO COLUMN** permite recuperar los parámetros de una columna existente.*

area es la referencia del área del informe rápido.

numColumna es el número de la columna a modificar.

titulo devuelve el título que será mostrado en el encabezado de la columna.

objeto devuelve el nombre del objeto de la columna (variable, campo o fórmula).

***Nota:** el comando no tiene en cuenta ninguna estructura virtual definida por medio de los comandos **SET TABLE TITLES** y **SET FIELD TITLES**. El nombre real del campo se devuelve en el parámetro objeto.*

oculta indica si la columna es visible o está oculta:

- *si oculta es igual a 1, la columna está oculta;*
- *si oculta es igual a 0, la columna es mostrada.*

tamaño devuelve el tamaño de la columna en píxeles. Si el valor devuelto es negativo, el tamaño de la columna es automático.

valoresRepetidos devuelve el estado de la propiedad de repetición de datos. Por ejemplo, si el valor de un campo o variable no cambia de un registro a otro, es posible repetirlo o no en cada línea de la columna.

• Si valoresRepetidos es igual a 0, los valores no se repiten.
• Si valoresRepetidos es igual a 1, los valores se repiten.

formato devuelve el formato de salida. Los formatos de salida son los formatos 4D compatibles con los datos mostrados en la columna.

Cuando se pasa, el parámetro opcional varResultado devuelve el nombre de la variable asignada automáticamente por el editor de Informes rápidos a la columna de la fórmula (si lo hay): "C1" para la primera columna de la fórmula, "C2" para la segunda y así sucesivamente. 4D utiliza esta variable para almacenar los resultados de la última ejecución de la fórmula de la columna cuando se genera el informe.

Informes tablas cruzadas

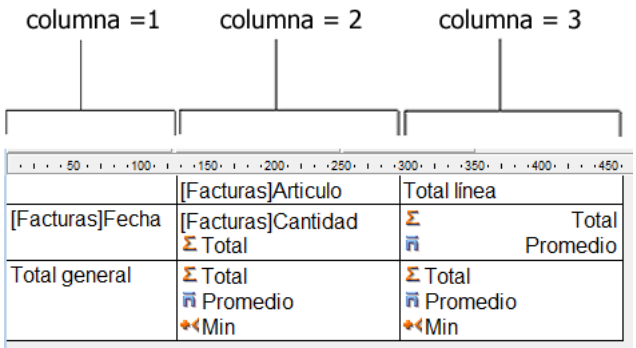
*Con este tipo de informe, el comando **QR GET INFO COLUMN** permite recuperar globalmente los mismos parámetros pero la referencia de las áreas a las cuales aplica es diferente y varia dependiendo del parámetro que quiera definir.*

Además, los parámetros titulo, oculta, y valoresRepetidos no se utilizan cuando este comando se utiliza en informes tablas cruzadas.

El valor a pasar en el parámetro numColumna depende de la operación que quiera efectuar de si quiere definir el tamaño de la columna o la fuente de datos y el formato de visualización.

- *Tamaño de la columna*

Este es un atributo "visual", por lo tanto las columnas son numeradas de izquierda a derecha, como se muestra en la siguiente imagen:



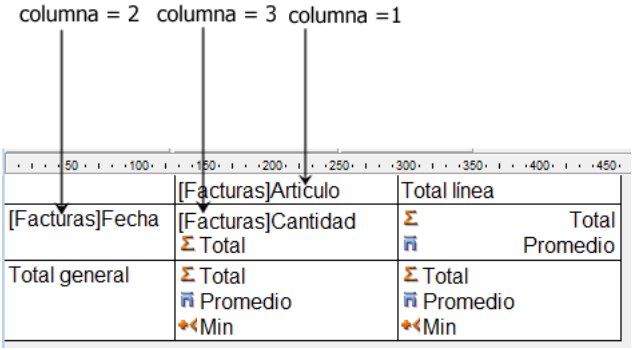
La siguiente instrucción define el tamaño automático para todas las columnas en un informe tabla cruzada y deja los otros elemento intactos:

```
For($i;1;3)
    QR GET INFO COLUMN(qr_area;$i;$titulo;$obj;$oculta;$tamaño;$rep;$format)
    QR SET INFO COLUMN(qr_area;$i;$titulo;$obj;$oculta;0;$rep;$format)
End for
```

*Notará que como quiere alterar únicamente el tamaño de la columna, tiene que utilizar **QR GET INFO COLUMN** para recuperar las propiedades de la columna y pasarlas a **QR SET INFO COLUMN** para dejarla intacta, excepto el tamaño de la columna.*

- *Fuente de datos (objeto) y formato de salida*

En este caso, la numeración de las columnas opera como se muestra a continuación:



Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro numColumna es incorrecto se genera el error -9852.

Ejemplo

Usted ha diseñado el siguiente informe:

	[People]LastName	[People]FirstName	C1	[People]City	[People]Salary
Title	LastName	FirstName	Age	City	Salary
Detail					
Grand total					

Puede escribir:

```
C_TEXT($vTitle;$vObject;$vDisplayFormat;$vResultVar)
C_LONGINT($area;$vHide;$vSize;$vRepeatedValue)
QR GET INFO COLUMN($area;3;$vTitle;$vObject;$vHide;$vSize;$vRepeatedValue;$vDisplayFormat;$vResultVar)
 //$vTitle = "Age"
 //$vObject = "[People]Birthdate-Current date"
 //$vHide = 0
 //$vSize = 57
 //$vRepeatedValue = 1
 //$vDisplayFormat = ""
 //$vResultVar = "C1"
```

QR Get info row

QR Get info row (area ; linea) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área creada
linea	Entero largo	➡	Línea
Resultado	Entero largo	↩	0 = visible, 1 = oculta

Descripción

*El comando **QR Get info row** indica si linea se muestra o se oculta en area.*
linea designa la línea a verificar:

- *si linea es igual a -1, verifica el título del informe*
- *si linea es igual a -2, verifica el área de detalle del informe*
- *si linea es igual a -3, verifica el área del total general*
- *si linea es un entero positivo, designa la línea de subtotal correspondiente.*

Puede utilizar las constantes del tema para designar el elemento de la línea (qr title= -1, qr detail=-2, qr grand total=-3)
El resultado de la función especifica si la línea está visible u oculta. Si es igual a 1, la línea está oculta; si es igual a 0, la línea es visible.
Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro linea es incorrecto, se genera el error -9852.

QR Get report kind

QR Get report kind (area) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	⇒	Referencia del área
Resultado	Entero largo	⇐	Tipo del informe

Descripción

El comando ***QR Get report kind*** *devuelve el tipo del informe presente en area.*

- *Si el comando devuelve 1, el informe es de tipo lista.*
- *Si el comando devuelve 2, el informe es de tipo tabla cruzada.*

Igualmente puede comparar el resultado de la función con las constantes del tema :

Constante	Tipo	Valor
qr cross report	Entero largo	2
qr list report	Entero largo	1

Si pasa un número de area incorrecto, se genera el error -9850.

QR Get report table

QR Get report table (area) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
Resultado	Entero largo	⮌	Número de tabla

Descripción

*El comando **QR Get report table** devuelve el número de la tabla actual del informe designado por el parámetro area.*
Si pasa un número de area inválido, se genera el error -9850.

QR GET SELECTION

QR GET SELECTION (area ; izquierda ; superior {; derecha {; inferior}})

Parámetro	*Tipo*		*Descripción*
area	*Entero largo*	➡	*Referencia del área*
izquierda	*Entero largo*	⬅	*Límite izquierdo*
superior	*Entero largo*	⬅	*Límite superior*
derecha	*Entero largo*	⬅	*Límite derecho*
inferior	*Entero largo*	⬅	*Límite inferior*

Descripción

El comando ***QR GET SELECTION*** *devuelve las coordenadas de la selección actual de area.*

izquierdo devuelve el número de la columna que es el límite izquierdo de la selección. Si izquierdo es igual a 0, toda la línea es seleccionada.

superior devuelve el número de la línea que es el límite superior de la selección. Si superior es igual a 0, toda la columna es seleccionada.

Nota: *si izquierdo y superior son iguales a 0, toda el área es selecionada.*

derecho es el número de la columna que es el límite derecho de la selección.

inferior es el número de la fila que es el límite superior de la selección.

Nota: *si no hay selección, los parámetros izquierdo, superior, derecho e inferior toman el valor -1.*

Si pasa un número de area inválido, se genera el error -9850.

QR GET SORTS

QR GET SORTS (area ; aColumnas ; aOrden)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
aColumnas	Array real	⬅	Columnas ordenadas
aOrden	Array real	⬅	Sentido de ordenación

Descripción

*El comando **QR GET SORTS** llenas dos arrays:*

- *aColumnas*
  Este array incluye todas las columnas que tienen un sentido de ordenación.
- *aOrden*
  Cada elemento de este array contiene el sentido de ordenación de la columna correspondiente.
  - Si aOrden{$i} es igual a 1, el sentido de la ordenación es ascendente.
  - Si aOrden{$i} es igual -1, el sentido de la ordenación es descendente.

Informes tablas cruzadas

En el caso de este tipo de informes, los arrays resultantes no pueden tener más de dos elementos ya que la ordenación sólo puede realizarse en las columnas (1) y las filas (2). (Valores para aColumnas).

Si pasa un número de area inválido, se genera el error -9850.

QR Get text property

QR Get text property (area ; numColumna ; numLinea ; propiedad) -> Resultado

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
numColumna	Entero largo	➡	Número de columna
numLinea	Entero largo	➡	Número de línea
propiedad	Entero largo	➡	Número de la propiedad
Resultado	Cadena, Entero largo	⬅	Valor de la propiedad seleccionada

Descripción

El comando **QR Get text property** *devuelve el valor de la propiedad de los atributos texto en la celda determinada por numColumna y numLinea.*

area es la referencia del área del informe rápido.

numColumna es el número de la columna de la celda.

numLinea es la referencia de la línea de la celda. Puede pasar:

- *un valor positivo designando el nivel de ruptura del subtotal correspondiente.*
- *una de las constantes del tema :*

Constante	Tipo	Valor	Comentario
qr detail	Entero largo	-2	Área de detalle del informe
qr footer	Entero largo	-5	Pie de página
qr grand total	Entero largo	-3	Área total general
qr header	Entero largo	-4	Encabezado de página
qr title	Entero largo	-1	Título del informe

Nota: *cuando pasa -4 ó -5 en numLinea, necesita pasar un número de columna en numColumna, incluso si no se utiliza.*

Nota: *en modo tabla cruzada, el principio es similar excepto para los valores de las líneas, que siempre son positivos.*

propiedad es el valor de la propiedad de texto a obtener. Puede utilizar las constantes del tema ***QR Propiedades de texto*** *y los siguientes valores pueden ser devueltos:*

Constante	Tipo	Valor	Comentario
_o_qr font	Entero largo	1	Obsoleto desde 4D v14R3 (utilice qr font name)
qr alternate background color	Entero largo	9	Número de color de fondo alterno
qr background color	Entero largo	8	Número de color de fondo
qr bold	Entero largo	3	Atributo negrita (0 ó 1)
qr font name	Entero largo	10	Nombre de la fuente como la devuelve por ejemplo el comando **FONT LIST**
qr font size	Entero largo	2	Tamaño de fuente expresado en puntos (9 a 255)
qr italic	Entero largo	4	Atributo itálica (0 ó 1)
qr justification	Entero largo	7	Atributo de justificación (0 por defecto, 1 a la izquierda, 2 al centro y 3 a la derecha)
qr text color	Entero largo	6	Atributo de número de color (Entero largo)
qr underline	Entero largo	5	Atributo de estilo subrayado (0 ó 1)

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro numColumna es incorrecto, se genera el error -9852.
Si el parámetro numLinea es incorrecto, se genera el error -9853.
Si el parámetro propiedad es incorrecto, se genera el error -9854

QR GET TOTALS DATA

QR GET TOTALS DATA (area ; numColumna ; numRuptura ; operador ; texto)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
numColumna	Entero largo	➡	Número de columna
numRuptura	Entero largo	➡	Número de ruptura
operador	Entero largo	⬅	Operador de la celda
texto	Cadena	⬅	Contenido de la celda

Descripción

Informe en lista

*El comando **QR GET TOTALS DATA** permite recuperar el contenido de una línea de ruptura específica.*

area es la referencia del área del informe rápido.

numColumna es el número de la columna de la celda cuyos datos serán recuperados.

numRuptura es el número de la línea de ruptura cuyos datos serán recuperados (subtotal o total general). Para una línea de subtotal, numRuptura corresponde al número de la línea. Para el total general, numRuptura vale -3 (también puede utilizar la constante qr grand total).

operador devuelve la suma de todos los operadores presentes en la celda. Puede utilizar las constantes del tema para tratar los valores devueltos:

Constante	Tipo	Valor
qr sum	Entero largo	1
qr average	Entero largo	2
qr min	Entero largo	4
qr max	Entero largo	8
qr count	Entero largo	16
qr standard deviation	Entero largo	32

Si operador devuelve 0, la celda no contiene ningún operador.

texto devuelve el texto en la celda.

***Nota:** operador y texto son mutuamente exclusivos, de manera que sólo uno de los dos parámetros devuelve un valor.*

Informe tabla cruzada

*El comando **QR GET TOTALS DATA** permite recuperar el contenido de una celda específica.*

area es la referencia del área del informe rápido.

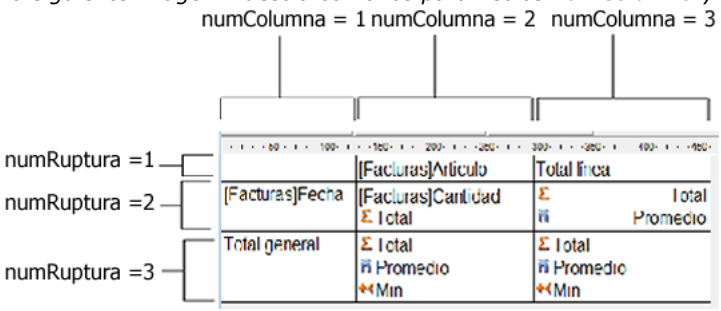
numColumna es el número de la columna de la celda cuyos datos van a ser recuperados.

numruptura es el número de la línea de la celda cuyos datos van a ser recuperados.

operador devuelve la suma de todos los operadores presentes en la celda. Puede utilizar las constantes del tema para procesar el valor devuelto (ver el párrafo anterior).

texto devuelve el texto en la celda.

La siguiente imagen muestra cómo los parámetros numColumna y numRuptura son combinados en una tabla cruzada:



Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro numColumna es incorrecto, se genera el error -9852.
Si el parámetro numRuptura es incorrecto, se genera el error -9853.

QR GET TOTALS SPACING

QR GET TOTALS SPACING (area ; subtotal ; valor)

Parámetro	*Tipo*		*Descripción*
area	*Entero largo*	➡	*Referencia del área*
subtotal	*Entero largo*	➡	*Número subtotal*
valor	*Entero largo*	⬅	*0=sin espacio, 32000=inserta un salto de página, >0=espacio añadido en la parte superior del nivel de ruptura, <0=aumento proporcional*

Descripción

*El comando **QR GET TOTALS SPACING** permite recuperar el valor del espacio añadido debajo de una línea de subtotal. Aplica únicamente en modo listado.*

area es la referencia del área del informe rápido.

subtotal es el nivel del subtotal (o nivel de ruptura) que se afectará. subtotal es un valor entre 1 y el número de líneas del subtotal/ruptura.

valor define el valor del espacio:

- *Si valor es igual a 0, no se añade ningún espacio.*
- *Si valor es igual a 32000, se inserta un salto de página.*
- *Si valor es un valor positivo, expresa el espacio a añadir en píxeles.*
- *Si valor es un valor negativo, expresa el espacio como un porcentaje de la línea de subtotal. Por ejemplo, el valor -100 indica un espacio debajo de la línea de subtotal correspondiente al 100% de la altura actual de la línea.*

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro subtotal es incorrecto, se genera el error -9852.

QR INSERT COLUMN

QR INSERT COLUMN (area ; numColumna ; objeto)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
numColumna	Entero largo	➡	Número de columna
objeto	Campo, Variable, Puntero	➡	Objeto a insertar en la columna

Descripción

*El comando **QR INSERT COLUMN** inserta o crea una columna en una posición específica. Las columnas situadas a la derecha de la columna añadida serán desplazadas en consecuencia.*

posicion es el número de la columna, establecida de izquierda a derecha.

El título por defecto de la columna será el valor pasado en objeto.

Si pasa un número de area inválido, se genera el error -9850.

***Nota:** este comando no puede ser utilizado con un informe tabla cruzada.*

Ejemplo

La siguiente instrucción inserta (o crea) una primera columna en el área MiArea, inserta "Campo1" como título de la columna (comportamiento por defecto) y llena el contenido del cuerpo con los valores del Campo1.

```
QR INSERT COLUMN(MiArea;1;->[Tabla 1]Campo1)
```

QR MOVE COLUMN

QR MOVE COLUMN (area ; numColumna ; nuevaPosicion)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
numColumna	Entero largo	➡	Número de la columna
nuevaPosicion	Entero largo	➡	Nueva posición de la columna

Descripción

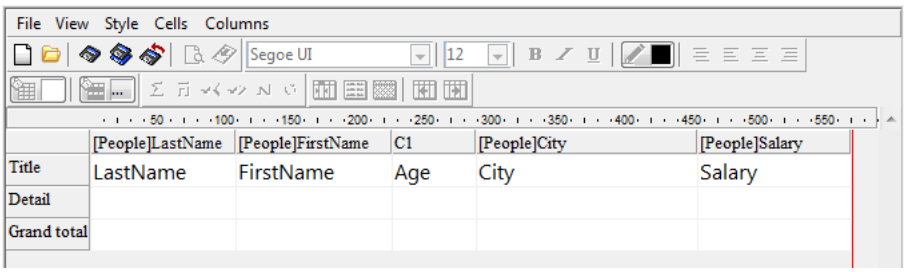
*El comando **QR MOVE COLUMN** mueve la columna que se encuentra actualmente en la posición numColumna a la posición nuevaPosicion.*

Tanto los parámetros numColumna y nuevaPosicion deben ser números válidos de columna (entre 1 y el número total de columnas en el informe); de lo contrario, se devuelve el error -9852.

***Nota:** este comando se puede usar solamente con los informes en lista.*

Ejemplo

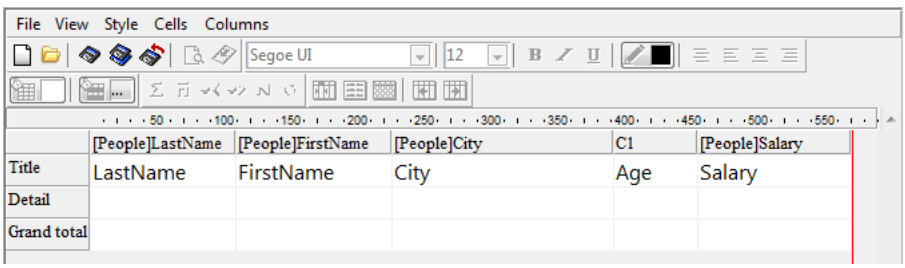
Usted diseñó el siguiente informe:



Si ejecuta:

```
QR MOVE COLUMN(area;3;4)
```

El resultado es:



QR NEW AREA

QR NEW AREA (ptr)

Parámetro	***Tipo***		***Descripción***
ptr	*Puntero*	⇒	*Puntero a una variable*

Descripción

*El comando **QR NEW AREA** crea una nueva área de informe rápido y almacena su número de referencia en la variable de tipo Entero largo referenciada por el puntero ptr.*

QR New offscreen area

QR New offscreen area -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entero largo*		*Referencia del área creada*

Descripción

*El comando **QR New offscreen area** crea un área de informe rápido fuera de pantalla y devuelve su número de referencia.*

QR ON COMMAND

QR ON COMMAND (area ; nomMetodo)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
nomMetodo	Cadena	➡	Nombre del método a llamar

Nota de compatibilidad

*A partir de 4D v16, se incluye un nuevo editor **Informes rápidos (64 bits)** con las versiones de 64 bits de 4D. La interfaz "ribbon" de este editor no es personalizable. Por lo tanto, en este contexto, **QR ON COMMAND** no se puede usar con la mayoría de las constantes (consulte también **QR EXECUTE COMMAND**).*

Descripción

*El comando **QR ON COMMAND** ejecuta el método proyecto 4D pasado en el parámetro nomMetodo cuando un comando del editor de informes rápidos es llamado por el usuario, seleccionando un comando de menú o haciendo clic en un botón.*

*Si area es igual a cero, nomMetodo se aplicará a todas las áreas del editor de informes rápidos hasta que se cierre la base o se ejecute la siguiente instrucción: **QR ON COMMAND**(0;"").*

nomMetodo recibe dos parámetros:

- *$1 es la referencia del área (Entero largo).*
- *$2 es el número del comando seleccionado (Entero largo). Puede comparar este valor con las constantes del tema **QR Comandos**.*

***Nota:** si planea compilar la base, es necesario declarar explícitamente los parámetros $1 y $2 como Enteros largos, incluso si no los utiliza.*

*Si quiere que el comando inicial se ejecute, debe incluir la siguiente instrucción en el método: **QR EXECUTE COMMAND**($1;$2).*

Si pasa un número de area inválido, se genera el error -9850.

QR REPORT

QR REPORT ({tabla ;} doc {; jerarquico {; asistente {; busqueda {; nomMetodo {; *}}}}})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a utilizar para el informe o tabla por defecto si se omite
doc	Cadena	➡	Documento de informe rápido a cargar
jerarquico	Booleano	➡	True = Mostrar las tablas Muchos relacionadas False o si se omite = No mostrar (por defecto)
asistente	Booleano	➡	True = Mostrar el botón del asistente False o si se omite = No mostrar (por defecto)
busqueda	Booleano	➡	True = Mostrar las herramientas de búsqueda y la tabla principal False o si se omite = No mostrar (por defecto)
nomMetodo	Cadena	➡	Nombre del método a llamar
*	Operador	➡	Eliminación de las cajas de diálogo de impresión

Descripción

QR REPORT *imprime un informe para tabla, con ayuda del editor de informes rápidos de 4D. Este editor permite a los usuarios crear sus propios informes. Para mayor información sobre la creación de informes rápidos con la ayuda del editor de informes rápidos, consulte la sección* ***Informes rápidos*** *o* ***Informes rápidos (64 bits)*** *en el Manual de Diseño de 4D.*

Notas:

- *El editor no aparece si la tabla ha sido declarada "Invisible."*
- *Cuando el editor se llama utilizando el comando* ***QR REPORT****, las relaciones entre las tablas conservan su estado manual, donde aplique. Este principio permite al desarrollador administrar él mismo este estado utilizando los comandos* ***SET AUTOMATIC RELATIONS*** *y* ***SET FIELD RELATION****. En versiones 32 bits, la opción* ***Todas las relaciones en automático*** *que se utiliza para modificar el estado automático/manual de las relaciones se oculta.*
- *El editor se llama en una ventana externa y no es posible utilizar el comando* ***QR ON COMMAND*** *en este contexto. Sin embargo, puede utilizar el parámetro nomMetodo para ejecutar código personalizado cuando un comando de interfaz se activa (ver abajo).*

El parámetro documento es un documento de informe que fue creado con el editor de informes rápidos y guardado en disco. El documento guarda las especificaciones del informe, no lo registros a imprimir. Si una cadena vacía ("") se pasa en documento, ***QR REPORT*** *muestra una caja de diálogo de apertura de archivos, en la cual el usuario puede seleccionar el informe a imprimir. Si el parámetro documento especifica un documento que no existe (por ejemplo, si pasa* ***Char(1)*** *en documento), se muestra el editor de informes rápidos.*

Versiones 32 bits únicamente:

- *El parámetro jerarquico indica si las tablas Muchos relacionadas se muestran en la lista de selección de campos. Por defecto, este valor es 0 (las tablas Muchos no se muestran).*
- *El parámetro asistente indica si el botón* ***Abrir el asistente*** *se va a mostrar en el editor de informes rápidos, permitiendo o no el acceso al asistente. Por defecto, este valor es False (no hay acceso al asistente).*
- *El parámetro busqueda indica si el botón Nueva búsqueda y el menú desplegable Tabla principal aparecerán en el editor de informes rápidos, por lo tanto permitiendo o no la modificación de la tabla actual y de la tabla principal actual. Por defecto, este valor es False (no hay acceso a las herramientas de búsqueda y a la tabla principal).*
- *El parámetro nomMetodo designa un método de proyecto 4D que se ejecuta cada vez que un comando del editor de informes rápidos es llamado por la selección de un elemento del menú o hacer clic en un botón. Utilizar este parámetro es equivalente a utilizar* ***QR ON COMMAND*** *en el contexto de la ventana del editor de informes rápidos (****QR ON COMMAND*** *sólo funciona en el contexto de un área incluida). Más en particular, se puede utilizar este nuevo parámetro para cambiar el juego de caracteres usado por el informe rápido.*
  El método nomMetodo recibe dos parámetros:
  - *$1 contiene la referencia del área (entero largo).*
  - *$2 contiene el número del comando seleccionado (entero largo). Puede comparar este valor con las constantes del tema* ***QR Comandos****.*

  Nota: *si desea compilar su base usando el compilador, debe declarar los parámetros $1 y $2 explícitamente como enteros largos, incluso si no los utiliza.*

  Si desea ejecutar el comando inicial elegido por el usuario, utilice la siguiente instrucción en el método nomMetodo:

```
QR EXECUTE COMMAND($1;$2)
```

  Si el parámetro nomMetodo es una cadena vacía ("") o se omite, ningún método se llama y se aplica la operación estándar de ***QR REPORT****.*

*Una vez seleccionado un informe, se muestran las cajas de diálogo de impresión, a menos que se especifique el parámetro *. Si se especifica este parámetro, no se muestran estas cajas de diálogo y se imprime el informe.*

Si no se muestra el editor de informes rápidos, la variable sistema OK toma el valor 1 si se imprime un informe; de lo contrario, toma el valor 0 (cero) (por ejemplo, si el usuario hace clic en ***Cancelar*** *en las cajas de diálogo de impresión).*

4D Server: *este comando puede ejecutarse en el servidor 4D Server en el marco de un procedimiento almacenado. En este contexto:*

- *Asegúrese de que no aparezca ninguna caja de diálogo en el equipo servidor (excepto para un requerimiento especifico). Para hacer esto, es necesario llamar al comando con el parámetro *.*
- *La sintaxis que hace aparecer el editor Quick Report no funciona con 4D Server; en este caso, la variable sistema OK toma el valor 0.*
- *En el caso de un problema relacionado con la impresora (sin papel, impresora desconectada, etc.), no se genera un error.*

Ejemplo 1

El siguiente ejemplo permite al usuario efectuar una búsqueda en la tabla [Personas], y luego imprime automáticamente el informe "Lista detallada":

```
QUERY([Personas])
If(OK=1)
   QR REPORT([Personas];"Detailed Listing";False;False;False;*)
End if
```

Ejemplo 2

El siguiente ejemplo permite al usuario efectuar una búsqueda en la tabla [Personas], y luego seleccionar el informe a imprimir:

```
QUERY([Personas])
If(OK=1)
   QR REPORT([Personas];"";False;False;False)
End if
```

Ejemplo 3

El siguiente ejemplo permite al usuario efectuar una búsqueda en la tabla [Personas], y luego muestra el editor de informes rápidos de manera que el usuario pueda diseña, guardar, cargar e imprimir informes con o sin el asistente:

```
QUERY([Personas])
If(OK=1)
   QR REPORT([Personas];Char(1);False;True)
End if
```

Ejemplo 4

*Consulte el ejemplo del comando **SET FIELD RELATION**.*

Ejemplo 5

*Usted desea convertir el conjunto de caracteres utilizado en un informe rápido llamado utilizando **QR REPORT** en Mac Roman:*

```
QR REPORT([MyTable];Char(1);False;False;False;"myCallbackMeth")
```

*El método **myCallbackMeth** convierte el informe cuando se genera:*

```
C_LONGINT($1;$2)
If($2=qr cmd generate) //si generamos un informe
   C_BLOB($myblob)
   C_TEXT($path;$text)
   C_LONGINT($type)
   QR EXECUTE COMMAND($1;$2) //ejecución del comando
   QR GET DESTINATION($1;$type;$path) //recuperación del destino
   If(($type=qr HTML file)|($type=qr text file))
      DOCUMENT TO BLOB($path;$myblob)
 //conversión del texto utilizando UTF-8
      $text:=Convert to text($myblob;"UTF-8")
 //uso del conjunto MacRoman
      CONVERT FROM TEXT($text;"MacRoman";$myblob)
 //Reenvío del informe convertido
      BLOB TO DOCUMENT($path;$myblob)
   End if
Else //de lo contrario, ejecución del comando
   QR EXECUTE COMMAND($1;$2)
End if
```

QR REPORT TO BLOB

QR REPORT TO BLOB (area ; BLOB)

Parámetro	Tipo		Descripción
area	Entero largo	⇒	Referencia del área
BLOB	BLOB	⇐	Blob a recibir el informe rápido

Descripción

El comando ***QR REPORT TO BLOB*** *coloca el informe cuya referencia se pasó en* area *en un BLOB (variable o campo).*
Si pasa un número de area *incorrecto, se genera el error -9850.*

Ejemplo

La siguiente instrucción asigna el informe rápido almacenado en el área MiArea a un campo BLOB.

```
QR REPORT TO BLOB(MiArea;[Tabla 1]Campo4)
```

QR RUN

QR RUN (area)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área a ejecutar

Descripción

*El comando **QR RUN** provoca la ejecución del informe rápido designado por el parámetro area. El informe se genera con sus parámetros actuales, incluyendo su tipo de salida. Puede utilizar el comando **QR SET DESTINATION** para modificar el tipo de salida.*

*El informe se ejecuta en la tabla a la que pertenece el área. Cuando area designa un área fuera de la pantalla, es necesario especificar la tabla a utilizar vía el comando **QR SET REPORT TABLE**.*

Si pasa un número de area inválido, se genera el error -9850.
***4D Server:** este comando puede ejecutarse en 4D Server como parte de un procedimiento almacenado. En este contexto, asegúrese de que no aparezca ninguna caja de diálogo en el equipo servidor (excepto para los requisitos específicos). Para ello, es necesario llamar al comando **QR SET DESTINATION** con el parámetro "*". En caso de un problema de la impresora (sin papel, impresora desconectada, etc), no se genera ningún mensaje de error.*

QR SET AREA PROPERTY

QR SET AREA PROPERTY (area ; propiedad ; valor)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
propiedad	Entero largo	➡	Elemento de interfaz
valor	Entero largo	➡	1 = mostrado, 0 = oculto

Descripción

*El comando **QR SET AREA PROPERTY** permite mostrar u ocultar el elemento de interfaz (barra de herramientas o barra de menús) cuya referencia se pasa en propiedad.*

La barra de menús y la barra de herramientas son numeradas del 1 al 6 (de arriba a abajo) y el valor 7 se asocia al menú contextual. Puede utilizar las constantes del tema para designar el elemento de interfaz:

Constante	Tipo	Valor	Comentario
qr view color toolbar	Entero largo	5	Muestra la barra de herramientas Colores (Mostrada=1, Oculta=0)
qr view column toolbar	Entero largo	6	Muestra la barra de herramientas Columnas (Mostrada=1, Oculta=0)
qr view contextual menus	Entero largo	7	Muestra los menús contextuales (Mostrados=1, Ocultos=0)
qr view menubar	Entero largo	1	Muestra la barra de menús (Mostrada=1, Oculta=0)
qr view operators toolbar	Entero largo	4	Muestra la barra de herramientas Operadores (Mostrada=1, Oculta=0)
qr view standard toolbar	Entero largo	2	Muestra la barra de herramientas estándar (Mostrada=1, Oculta=0)
qr view style toolbar	Entero largo	3	Muestra la barra de herramientas Estilo (Mostrada=1, Oculta=0)

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro propiedad es incorrecto, se genera el error -9852.

QR SET BORDERS

QR SET BORDERS (area ; columna ; linea ; borde ; grueso {; color})

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
columna	Entero largo	➡	Número de columna
linea	Entero largo	➡	Número de línea
borde	Entero largo	➡	Valor del borde compuesto
grueso	Entero largo	➡	Ancho de la línea
color	Entero largo	➡	Color del borde

Descripción

*El comando **QR SET BORDERS** permite definir el estilo del borde de una celda dada.*

area es la referencia del área del informe rápido.

columna es el número de la columna de la celda.

linea es el número de línea de la celda.
- *si linea es igual a -1, el título del informe se afecta*
- *si linea es igual a -2, el detalle del informe se afecta*
- *si linea es igual a -3, el total general del informe se afecta*
- *si linea es un entero positivo, designa la línea del subtotal correspondiente.*

Puede utilizar las constantes del tema para designar el elemento de la línea (qr title= -1, qr detail=-2, qr grand total=-3).

borde es un valor compuesto que indica el o los bordes de la celda a modificar:

- *1 indica el borde de izquierdo*
- *2 indica el borde superior*
- *4 indica el borde derecho*
- *8 indica el borde inferior*
- *16 indica el borde interior vertical*
- *32 indica el borde interior horizontal.*

Puede utilizar las constantes del tema para designar el elemento del borde.

Por ejemplo, si pasa 5 en borde los bordes izquierdo y derecho se afectarán.

ancho es al ancho de la línea:

- *0 indica que no hay línea*
- *1 indica un ancho de línea de 1/4 punto*
- *2 indica un ancho de línea de 1/2 punto*
- *3 indica un ancho de línea de 1 punto*
- *4 indica un ancho de línea de 2 puntos*

color es el color de la línea:

- *Si color es un valor positivo, indica un color específico.*
- *Si color es igual a 0, el color es negro.*
- *Si color es igual a -1, el color no cambia.*

***Nota:** el color por defecto es el negro.*

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro columna es incorrecto, se genera el error -9852.
Si el parámetro linea es incorrecto, se genera el error -9853.
Si el parámetro borde es incorrecto, se genera el error -9854.
Si el parámetro ancho es incorrecto, se genera el error -9855.

QR SET DESTINATION

QR SET DESTINATION (area ; tipo {; especificos})

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
tipo	Entero largo	➡	Tipo de informe
especificos	Cadena, Variable	➡	Específicos del tipo de salida

Descripción

El comando **QR SET DESTINATION** permite definir el tipo de salida del informe rápido en area.

En el parámetro tipo, pase una de las constantes del tema **QR Destino de salida**. El contenido del parámetro especificos depende del valor de tipo. La siguiente tabla describe los valores que pueden pasarse en los parámetros tipo y especificos:

Constante	Tipo	Valor	Comentario
_o_qr 4D Chart area	Entero largo	4	*** Constante obsoleta ***
qr 4D View area	Entero largo	3	detalles: N.A.
qr HTML file	Entero largo	5	detalles: ruta de acceso al archivo.
qr printer	Entero largo	1	detalles: "*" para eliminar las cajas de diálogo de impresión
qr text file	Entero largo	2	detalles: ruta de acceso al archivo.

qr printer (1): si pasa una cadena que contiene un asterisco ("*") en el parámetro especificos, no se mostrará una caja de diálogo durante la impresión, los parámetros de impresión actuales se utilizarán automáticamente. Esta configuración es necesaria si desea imprimir el reporte en el servidor.

qr text file (2): si pasa una cadena vacía como en el parámetro especificos, se muestra una caja de diálogo estándar de guardar archivo. Si pasa una ruta de acceso valida, el informe rápido se guardará en la ubicación indicada.

Por defecto, el delimitador del campo es el carácter tab (ASCII 9). El delimitador de registro por defecto es el carácter retorno de carro (ASCII 13). Puede cambiar estos caracteres por defecto asignando valores a las variables sistema: FldDelimit y RecDelimit. Si en Windows, FldDelimit es igual a 13, un carácter 10 (salto de de línea) será añadido después del retorno de carro. Tenga en cuenta que estas variables son utilizadas por otros comandos como por ejemplo **IMPORT TEXT**. Toda modificación de estas variables repercute en toda la aplicación.

qr 4D View area (3): si el usuario tiene 4D View, se crea una ventana externa 4D View y muestra los resultados de los parámetros actuales del área del informe rápido.

qr HTML file (5): se crea un archivo HTML utilizando la plantilla definida por **QR SET HTML TEMPLATE**. Para mayor información sobre el modo de conversión de los datos, consulte el manual de Diseño.

Si pasa un número de area inválido, se genera el error -9850.
Si el valor del destino tipo es incorrecto, se genera el error -9852.

Ejemplo

El siguiente código define el archivo texto Midoc.txt como tipo de destino del informe rápido y lo ejecuta:

```
QR SET DESTINATION(MiArea;qr text file;"MiDoc.txt")
QR RUN(MiArea)
```

QR SET DOCUMENT PROPERTY

QR SET DOCUMENT PROPERTY (area ; propiedad ; valor)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
propiedad	Entero largo	➡	1 = Diálogo de impresión, 2 = Unidad del documento
valor	Entero largo	➡	Valor de la propiedad

Descripción

El comando ***QR SET DOCUMENT PROPERTY*** *permite mostrar la caja de diálogo de impresión o definir la unidad utilizada por el documento.*

En propiedad, puede pasar unas de las constantes del tema ***QR Propiedades de documento****:*

Constante	Tipo	Valor	Comentario
qr printing dialog	Entero largo	1	*Visualización de la caja de diálogo de impresión:* • *Si valor = 0, la caja de diálogo de impresión no se muestra antes de la impresión.* • *Si valor = 1, la caja de diálogo de impresión se muestra antes de la impresión (valor por defecto).*
qr unit	Entero largo	2	*Unidad del documento:* • *Si valor = 0, la unidad del documento es el punto.* • *Si valor = 1, la unidad del documento es el centímetro.* • *Si valor = 2, la unidad del documento es la pulgada.*

- *Si propiedad es igual a 1, el comando aplica a la visualización del diálogo de impresión.*

  - Si valor es igual a 1, la caja de diálogo de impresión se muestra antes de la impresión (valor por defecto).
  - Si valor es igual a 0, la caja de impresión no se muestra antes de la impresión.

- *Si propiedad es igual a 2, el comando se aplica a la unidad del documento.*

  - Si valor es igual a 0, la unidad del documento es el punto.
  - Si valor es igual a 1, la unidad del documento es el centímetro.
  - Si valor es igual a 2, la unidad del documento es la pulgada.

Si pasa un número de area incorrecto, se genera el error -9850.
Si pasa un valor incorrecto en propiedad o valor, se genera el error correspondiente (-9852 o -9853).

QR SET HEADER AND FOOTER

QR SET HEADER AND FOOTER (area ; selector ; tituloIzq ; tituloCent ; tituloDer ; altura {; imagen {; alinImag}})

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
selector	Entero largo	➡	1 = Encabezado, 2 = Pie de página
tituloIzq	Cadena	➡	Texto mostrado a la izquierda
tituloCent	Cadena	➡	Texto mostrado en el centro
tituloDer	Cadena	➡	Texto mostrado a la derecha
altura	Entero largo	➡	Altura del encabezado o pie de página
imagen	Imagen	➡	Imagen a mostrar
alinImag	Entero largo	➡	Atributo de alineación de la imagen

Descripción

*El comando **QR SET HEADER AND FOOTER** permite definir el contenido y el tamaño del encabezado y el pie de página de area. selector le permite seleccionar el encabezado o pie de página:*

- *si selector es igual a 1, se afectará el encabezado;*
- *si selector es igual a 2, se afectará el pie de página.*

tituloIzq, tituloCent y tituloDer son los valores para los encabezados/pies de páginas izquierdo, centro y derecha, respectivamente.
altura es la altura del encabezado/pie de página, expresada en la unidad seleccionada para el informe rápido.
imagen contiene la imagen a mostrar en el encabezado o pie de página.
alinImag es el atributo de alineación para la imagen pasada en imagen.

- *Si alinImag es igual a 1, la imagen se alinea a la izquierda.*
- *Si alinImag es igual a 2, la imagen se centra.*
- *Si alinImag es igual a 3, la imagen se alinea a la derecha.*

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro selector es incorrecto, se genera el error -9852.

Ejemplo

La siguiente instrucción coloca el título "Título del centro" en el encabezado del informe rápido en MiArea y define la altura del encabezado en 200 puntos:

```
QR SET HEADER AND FOOTER(MiArea;1;"";"Título del centro";"";200)
```

QR SET HTML TEMPLATE

QR SET HTML TEMPLATE (area ; plantilla)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
plantilla	Texto	➡	Código de la plantilla HTML

Descripción

*El comando **QR SET HTML TEMPLATE** define la plantilla HTML a utilizar para el área de informe rápido referenciada por area. Esta plantilla se utilizará durante la creación del informe en formato HTML.*

La plantilla utiliza un conjunto de etiquetas para procesar los datos. Este funcionamiento permite generar documentos HTML cercanos al informe original o documentos con apariencia totalmente personalizada.

***Nota:** primero debe llamar **QR SET DESTINATION** para definir el formato HTML como destino de salida.*

Etiquetas HTML

<!--#4DQRheader--> ... <!--/#4DQRheader-->
Los títulos de las columnas se insertarán entre las etiquetas. Estas etiquetas generalmente son utilizadas para definir la línea del título del informe.

<!--#4DQRrow--> ... <!--/#4DQRrow-->
La información insertada entre estas etiquetas se repite para cada línea de datos (incluyendo las líneas de detalle y subtotal).

<!--#4DQRcol--> ... <!--/#4DQRcol-->
La información insertada entre estas etiquetas se repite para cada columna de datos dentro de una línea. El orden de la columna permanecerá idéntico al orden en el informe. Cuando se utilizan conjuntamente con <!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->, las etiquetas <!--#4DQRcol--> ... <!--/#4DQRcol--> no serán efectivas a través de las columnas cuyos contenidos no son insertados utilizando <!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->.

Por ejemplo, en un informe que tiene cinco columnas, usted utiliza las etiquetas <!--#4DQRcol;2--> ... <!--/#4DQRcol;2--> para insertar los datos de la segunda columna, <!--#4DQRcol--> ... <!--/#4DQRcol--> irán por cada fila, a través de las columnas 1, 3, 4, y 5. Estas últimas etiquetas ignoran la columna cuyo contenido se publica utilizando <!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->.

<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->
La información insertada entre estas etiquetas se extrae de la columna del informe cuyo número es "n". Si quiere mostrar por ejemplo las columnas en un orden diferente al del informe inicial, puede escribir:
<!--#4DQRrow--> <!--#4DQRcol;3--> ... <!--/#4DQRcol;3--><!--#4DQRcol;2--> ... <!--/#4DQRcol;2--><!--#4DQRcol;1--> ... <!--/#4DQRcol;1--> <!--/#4DQRrow-->

En este ejemplo, las columnas se insertan en el orden inverso del informe.

<!--#4DQRfont--> ... <!--/#4DQRfont-->
El contenido HTML insertado entre estas etiquetas será utilizada para la definición de la fuente de la columna o celda actual.

<!--#4DQRfont--> se reemplazará por una definición de fuente HTML y <!--/#4DQRfont--> se reemplazará por la etiqueta de cierre estándar ().

<!--#4DQRface--> ... <!--/#4DQRface-->
El contenido HTML insertado entre estas etiquetas será utilizará para la definición del estilo de la columna o celda actual.

<!--#4DQRface--> se reemplazará por una definición de estilo HTML <!--#4DQRface--> se reemplazará por la etiqueta de cierre estándar (</face>).

<!--#4DQRbgcolor-->
Esta etiqueta de color se reemplazará por la definición de color de la celda actual.

<!--#4DQRdata-->
Esta etiqueta se reemplazará por los datos de la celda actual.

<!--#4DQRlHeader--><!--#4DQRdata--><!--/#4DQRlHeader-->
<!--#4DQRcHeader--><!--#4DQRdata--><!--/#4DQRcHeader-->
<!--#4DQRrHeader--><!--#4DQRdata--><!--/#4DQRrHeader-->
Esas etiquetas se reemplazarán respectivamente por los datos en el encabezado de la izquierda, centro y derecha.

<!--#4DQRlFooter--><!--#4DQRdata--><!--/#4DQRlFooter-->
<!--#4DQRcFooter--><!--#4DQRdata--><!--/#4DQRcFooter-->
<!--#4DQRrFooter--><!--#4DQRdata--><!--/#4DQRrFooter-->

Estas etiquetas se reemplazarán respectivamente por los datos del pie de página de la izquierda, centro o derecha.

Si pasa un número de area inválido, se genera el error -9850.

QR SET INFO COLUMN

QR SET INFO COLUMN (area ; numColumna ; titulo ; objeto ; oculta ; tamaño ; valoresRepetidos ; formato)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
numColumna	Entero largo	➡	Número de columna
titulo	Cadena	➡	Título de la columna
objeto	Campo, Variable	➡	Objeto asignado a la columna
oculta	Entero largo	➡	0 = visible, 1 = oculto
tamaño	Entero largo	➡	Largo de la columna
valoresRepetidos	Entero largo	➡	0 = no repetidos, 1 = repetido
formato	Cadena	➡	Formato de visualización

Descripción

Informes en lista

*El comando **QR SET INFO COLUMN** permite definir los parámetros de una columna existente.*

area es la referencia del área del informe rápido.

numColumna es el número de la columna a definir.

titulo es el título que será mostrado en el encabezado de la columna.

objeto es el objeto de la columna (variable, campo o fórmula).

oculta especifica si la columna es visible o está oculta:

- *si oculta es igual a 1, la columna está oculta;*
- *si oculta es igual a 0, la columna es mostrada.*

tamaño es el tamaño en píxeles a asignar a la columna. Si tamaño es igual a -1, el tamaño de la columna es automático.

valoresRepetidos indica el estado de la propiedad de repetición de datos. Por ejemplo, si el valor de un campo o variable no cambia de un registro a otro, es posible repetirlo o no en cada línea de la columna.

• Si valoresRepetidos es igual a 0, los valores no se repiten.
• Si valoresRepetidos es igual a 1, los valores se repiten.

formato es el formato de salida. Los formatos de salida son los formatos 4D compatibles con los datos mostrados en la columna.

La siguiente instrucción define el título para la columna #1, el contenido del Campo2, hace que la columna sea visible con un ancho de 150 píxeles y define el formato de salida ###.##.

```
QR SET INFO COLUMN(area;1;"Titulo";"[Tabla 1]Campo2";0;150;0;"###,##")
```

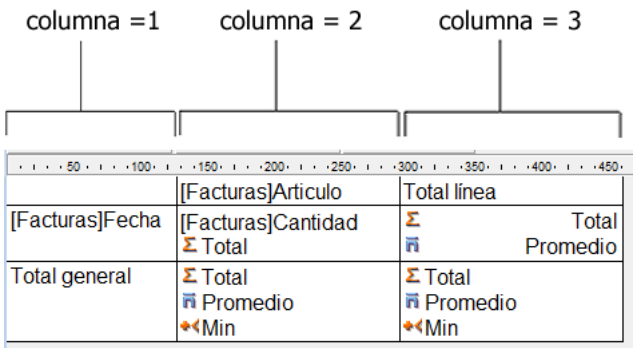
Informes tablas cruzadas

*El comando **QR SET INFO COLUMN** permite definir globalmente los mismos parámetros pero la referencia de las áreas a las cuales aplica es diferente y varia dependiendo del parámetro que quiera definir.*

Además, los parámetros titulo, oculta, y valoresRepetidos no se utilizan cuando este comando se utiliza en informes tablas cruzadas. El valor a utilizar en numColumna varia dependiendo de si quiere definir el tamaño de la columna o la fuente de datos y el formato de visualización.

- *Tamaño de la columna*

Este es un atributo "visual", por lo tanto las columnas son numeradas de izquierda a derecha, como se muestra en la siguiente imagen:



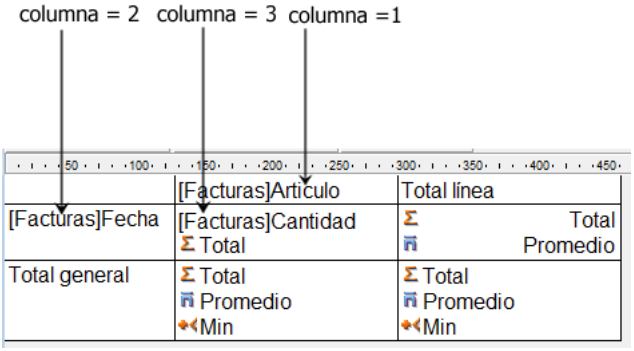
El siguiente método define el tamaño automático para todas las columnas en informe tablas cruzadas y deja los otros elementos intactos:

```
For($i;1;3)
   QR GET INFO COLUMN(qr_area;$i;$titulo;$obj;$oculta;$tamaño;$rep;$format)
   QR SET INFO COLUMN(qr_area;$i;$titulo;$obj;$oculta;0;$rep;$format)
End for
```

*Notará que como quiere modificar únicamente el tamaño de la columna, debe utilizar el comando **QR GET INFO COLUMN** para recuperar las propiedades actuales de la columna y pasarla a **QR SET INFO COLUMN** con el fin de conservarla intacta, excepto por el tamaño de la columna.*

- *Fuente de datos (objeto) y formato de visualización*

En este caso la numeración de las columnas se efectúa de la siguiente manera:

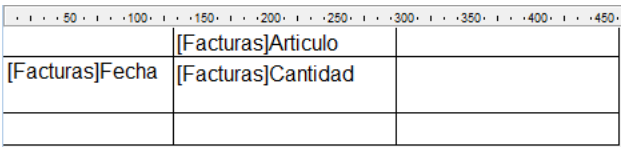


*Notará que no es posible direccionar todas las celdas utilizando el comando **QR SET INFO COLUMN**, las celdas que no son numeradas arriba son direccionadas utilizando **QR SET TOTALS DATA**.*

El siguiente código asigna las fuentes de datos a las tres celdas necesarias para la creación de un informe tabla cruzada simple:

```
QR SET REPORT TABLE(qr_area;Table(->[Facturas]))
ALL RECORDS([Facturas])
QR SET REPORT KIND(qr_area;2)
QR SET INFO COLUMN(qr_area;1;"";->[Facturas]Elemento;1;-1;1;"")
QR SET INFO COLUMN(qr_area;2;"";->[Facturas]Fecha;1;-1;1;"")
QR SET INFO COLUMN(qr_area;3;"";->[Facturas]Cantidad;1;-1;1;"")
```

Se genera la siguiente área de informe:



Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro numColumna es incorrecto, se genera el error -9852.

QR SET INFO ROW

QR SET INFO ROW (area ; linea ; oculta)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
linea	Entero largo	➡	Línea
oculta	Entero largo	➡	0 = visible, 1 = oculta

Descripción

*El comando **QR SET INFO ROW** muestra/oculta la fila cuya referencia se pasa en linea.*
linea designa la línea a modificar:

- *si línea es igual a -1, el título del informe se afecta,*
- *si línea es igual a -2, el detalle del informe se afecta,*
- *si línea es igual a -3, el total general del informe se afecta,*
- *si línea es un entero positivo, designa la línea del subtotal correspondiente.*

Puede utilizar las constantes del tema para designar el elemento de línea (qr title= -1, qr detail=-2, qr grand total=-3).
oculta especifica si la línea está visible u oculta:

- *si oculta es igual a 1, la línea está oculta;*
- *si oculta es igual a 0, la línea es visible.*

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro linea es incorrecto, se genera el error -9852.

Ejemplo

La siguiente instrucción oculta el contenido de la línea detalle:

```
QR SET INFO ROW(area;qr detail;1)
```

QR SET REPORT KIND

QR SET REPORT KIND (area ; tipo)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
tipo	Entero largo	➡	Tipo del informe

Descripción

El comando **QR SET REPORT KIND** define el tipo del informe presente en area.

- Si tipo es igual a 1, el informe es de tipo lista.
- Si tipo es igual a 2, el informe es de tipo tabla cruzada.

Igualmente puede utilizar las constantes del tema :

Constante	Tipo	Valor
qr cross report	Entero largo	2
qr list report	Entero largo	1

Si define un nuevo tipo para un informe existente, se eliminan los parámetros anteriores y se crea un nuevo informe vacío.
Si pasa un número de area inválido, se genera el error -9850.
Si pasa un valor incorrecto del parámetro propiedad, se genera el error -9852.

QR SET REPORT TABLE

QR SET REPORT TABLE (area ; tabla)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
tabla	Entero largo	➡	Número de tabla

Descripción

*El comando **QR SET REPORT TABLE** define la tabla actual para el área del informe referenciada por el parámetro area a la tabla número tabla.*

Es necesario que una tabla sea asignada al informe ya que el editor de informes utilizará la selección actual de esta tabla para mostrar los datos, efectuar los cálculos y propagar relaciones, si es necesario.

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro tabla es incorrecto, se genera el error -9852.

QR SET SELECTION

QR SET SELECTION (area ; izquierda ; superior {; derecha {; inferior}})

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
izquierda	Entero largo	➡	Límite izquierdo
superior	Entero largo	➡	Límite superior
derecha	Entero largo	➡	Límite derecho
inferior	Entero largo	➡	Límite inferior

Descripción

*El comando **QR SET SELECTION** permite seleccionar una celda, una fila, una columna o toda un area como lo haría con un ratón. Este comando también le permite deseleccionar la selección actual.*

izquierdo es el número del límite izquierdo. Si izquierdo es igual a 0, toda la línea es seleccionada.
superior es el número del límite superior. Si superior es igual a 0, toda la columna es seleccionada.
derecho es el número del límite derecho.
inferior es el número del límite inferior.

Notas:
- *Si izquierdo y superior son iguales a 0, toda el área es seleccionada.*
- *Para deseleccionar todo, pase -1 en izquierdo, derecho, superior e inferior.*

Si pasa un número de area inválido, se genera el error -9850.

QR SET SORTS

QR SET SORTS (area ; aColumnas {; aOrden})

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
aColumnas	Array real	➡	Columnas
aOrden	Array real	➡	Sentido de ordenación

Descripción

*El comando **QR SET SORTS** permite definir el sentido de ordenación de cada columna del informe rápido cuya referencia se pasa en area.*

aColumnas: debe almacenar en este array el número de cada columna para la cual quiere definir un sentido de ordenación.

aOrden: cada elemento de este array debe contener el sentido de ordenación para la columna correspondiente referenciada en el array aColumnas.

- *Si aOrden{$i} es igual a 1, el sentido de la ordenación es ascendente.*
- *Si aOrden{$i} es igual a - 1, el sentido de la ordenación es descendiente.*

Informes tablas cruzadas

En el caso de este tipo de informes, el array no puede tener más de dos elementos. Únicamente puede ordenar las columnas (1) y las filas (2). Los datos (situados en la intersección de las columnas y las líneas) no pueden ordenarse con este comando.

Este es el código para ordenar sólo las líneas en caso de un informe de tablas cruzadas:

```
ARRAY REAL($aColumnas;1)
$aColumnas{1}:=2
ARRAY REAL($aOrdenes;1)
$aOrden{1}:=-1  `Orden alfabético de las líneas
QR SET SORTS(qr_area;$aColumnas;$aOrden)
```

Si pasa un número de area inválido, se genera el error -9850.

QR SET TEXT PROPERTY

QR SET TEXT PROPERTY (area ; numColumna ; numLinea ; propiedad ; valor)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
numColumna	Entero largo	➡	Número de columna
numLinea	Entero largo	➡	Número de línea
propiedad	Entero largo	➡	Número de propiedad
valor	Entero largo, Cadena	➡	Valor de la propiedad seleccionada

Descripción

El comando ***QR SET TEXT PROPERTY*** *permite definir las propiedades de texto de la celda determinada por los parámetros numColumna y numLinea.*

area es la referencia del área del informe rápido.

numColumna es el número de la columna de la celda.

numLinea es la referencia de la línea de la celda. Puede pasar:

- *un valor positivo, designando la línea del subtotal correspondiente.*
- *una de las constantes del tema* ***QR Filas para propiedades****:*

Constante	Tipo	Valor	Comentario
qr detail	Entero largo	-2	Área de detalle del informe
qr footer	Entero largo	-5	Pie de página
qr grand total	Entero largo	-3	Área total general
qr header	Entero largo	-4	Encabezado de página
qr title	Entero largo	-1	Título del informe

Nota: *cuando pasa -4 o -5 en numLinea, necesita pasar un número de columna en numColumna, incluso si no lo utiliza.*

Nota: *en modo tablas cruzadas, el principio es similar excepto para los valores de las líneas, que siempre son positivos.*

propiedad es el valor del atributo de texto a asignar. Puede utilizar las constantes del tema ***QR Propiedades de texto*** *y los siguientes valores pueden definirs*

Constante	Tipo	Valor	Comentario
_o_qr font	Entero largo	1	Obsoleto desde 4D v14R3 (utilice qr font name)
qr alternate background color	Entero largo	9	Número de color de fondo alterno
qr background color	Entero largo	8	Número de color de fondo
qr bold	Entero largo	3	Atributo negrita (0 ó 1)
qr font name	Entero largo	10	Nombre de la fuente como la devuelve por ejemplo el comando ***FONT LIST***
qr font size	Entero largo	2	Tamaño de fuente expresado en puntos (9 a 255)
qr italic	Entero largo	4	Atributo itálica (0 ó 1)
qr justification	Entero largo	7	Atributo de justificación (0 por defecto, 1 a la izquierda, 2 al centro y 3 a la derecha)
qr text color	Entero largo	6	Atributo de número de color (Entero largo)
qr underline	Entero largo	5	Atributo de estilo subrayado (0 ó 1)

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro numColumna incorrecto, se genera el error -9852.
Si el parámetro numLinea incorrecto, se genera el error -9853.
Si el parámetro propiedad incorrecto, se genera el error -9854.

Ejemplo

Este método define varios atributos para el título de la primera columna:

```
 //Asigna la fuente Times:
QR SET TEXT PROPERTY(qr_area;1;-1;qr font name;"Times")
 //asigna el tamaño de fuente de 10 puntos:
QR SET TEXT PROPERTY(qr_area;1;-1;qr font size;10)
 //asigna el atributo de fuente negrita:
QR SET TEXT PROPERTY(qr_area;1;-1;qr bold;1)
 //asigna el atributo de fuente Italica:
QR SET TEXT PROPERTY(qr_area;1;-1;qr italic;1)
```

```
 //asigna el atributo de fuente subrayado:
QR SET TEXT PROPERTY(qr_area;1;-1;qr underline;1)
 //asigna el color verde claro:
QR SET TEXT PROPERTY(qr_area;1;-1;qr text color;0x0000FF00)
```

QR SET TOTALS DATA

QR SET TOTALS DATA (area ; numColumna ; numRuptura ; operador | valor)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
numColumna	Entero largo	➡	Número de columna
numRuptura	Entero largo	➡	Número de ruptura
operador | valor	Entero largo, Cadena	➡	Operador para la celda o contenido de la celda

Descripción

Nota: *este comando no puede crear un subtotal.*

Modo listado

El comando ***QR SET TOTALS DATA*** *permite definir el contenido de una línea de ruptura específica (total o subtotal).*

area es la referencia del área del informe rápido.

numColumna es el número de columna de la celda que quiere definir.

numRuptura es el número de la línea de ruptura a modificar (subtotal o total general). Para una línea de subtotal, numRuptura corresponde al número del orden de la ruptura. Para el total general, numRuptura es igual a -3 o la constante qr grand total.

operador es el valor acumulado de todos los operadores presentes en la celda. Utilice las constantes del tema ***QR Operadores*** *para definir este parámetro:*

ConstanteValor

qr sum1
qr average2
qr min4
qr max8
qr count16
qr standard deviation32

Si operador es igual a 0, no hay operador.

valor es el texto a ubicar en la celda.

Nota: *operador/valor son mutuamente exclusivos, de manera que puede definir un operador o un texto.*

Puede pasar los siguientes valores:
- # para el valor que provocó la ruptura o el subtotal.
- ##S se reemplazará por la suma.
- ##A se reemplazará por el promedio.
- ##C se reemplazará por el número
- ##X se reemplazará por el máximo.
- ##N se reemplazará por el mínimo.
- ##D se reemplazará por la desviación estándar.
- ##xx, donde xx es un número de columna. Este código se reemplazará por el valor de la columna, utilizando su propio formato. Si esta columna no existe, entonces no se reemplazará.

Informe tabla cruzada

El comando ***QR SET TOTALS DATA*** *permite definir el contenido de una celda específica.*

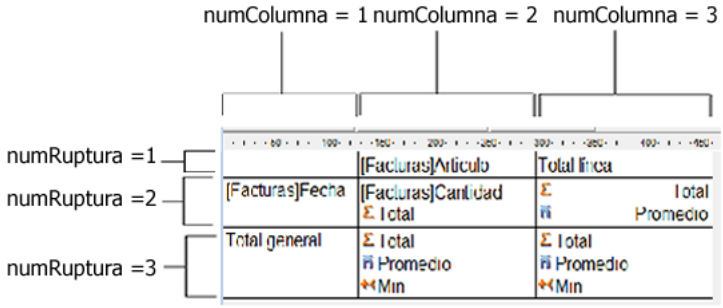
area es la referencia del área del informe rápido.

numColumna es el número de columna de la celda que se va a definir.

numRuptura es el número de la línea de la celda que se va a definir.

operador contiene el valor acumulado de todos los operadores presentes en la celda. Puede utilizar las constantes del tema ***QR Operators*** *para definir este parámetro (ver el párrafo anterior).*
valor es el texto a ubicar en la celda.

La siguiente imagen muestra cómo los parámetros numColumna y numRuptura son combinados en modo tabla cruzada:



Tipos de datos soportados

Puede pasar dos tipos de datos:

- *Título*
  Un título se pasa vía el parámetro valor. Este valor es una cadena y puede pasarse únicamente con las siguientes celdas: colNum=3 breakNum=1 y colNum=1 breakNum=3.

- *Operador*
  Un operador o una combinación de operadores (como se describió anteriormente) puede ser pasado por las siguientes celdas:
  numColumna=2, numRuptura=2
  numColumna=3, numRuptura=2
  numColumna=2, numRuptura=3

Note que estos dos últimos valores afectan igualmente a la celda (Columna 3; Línea 3). Si por ejemplo un cálculo se efectúa en la celda (Columna 2; Línea 3), el contenido de la celda (Columna 3; Línea 3) será modificado en consecuencia.

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro numColumna es incorrecto, se genera el error -9852.
Si el parámetro numRuptura es incorrecto, se genera el error -9853.

QR SET TOTALS SPACING

QR SET TOTALS SPACING (area ; subtotal ; valor)

Parámetro	Tipo		Descripción
area	Entero largo	➡	Referencia del área
subtotal	Entero largo	➡	Número del subtotal
valor	Entero largo	➡	0=sin espacio, 32000=inserta un salto de página, >0=espacio añadido en la parte superior del nivel de ruptura, <0=aumento proporcional

Descripción

El comando ***QR SET TOTALS SPACING*** *define un espacio debajo de la línea de subtotal. Aplica únicamente al modo listado.*
area es la referencia del área del informe rápido.
subtotal es el nivel del subtotal (o de ruptura) a modificar.
valor define el valor del espacio:

- *Si valor es igual a 0, no se añade ningún espacio.*
- *Si valor es igual a 32000, se añade un salto de página.*
- *Si valor es un valor positivo, expresa el espacio a añadir en píxeles.*
- *Si valor es un valor negativo, expresa el espacio como un porcentaje de la línea de subtotal. Por ejemplo, el valor -100 definirá un espacio debajo de la línea del subtotal correspondiente al 100% de la altura de la línea.*

Nota: *si el espacio debajo de la línea de subtotal "empuja" a la línea a la siguiente página, no se insertará espacio sobre la línea en esa página.*

Si pasa un número de area inválido, se genera el error -9850.
Si el parámetro subtotal, es incorrecto, se genera el error -9852.

Interfaz de usuario

- *BEEP*
- *Caps lock down*
- *Focus object*
- *GET FIELD TITLES*
- *GET MOUSE*
- *GET TABLE TITLES*
- *HIDE MENU BAR*
- *Macintosh command down*
- *Macintosh control down*
- *Macintosh option down*
- *PLAY*
- *Pop up menu*
- *POST CLICK*
- *POST EVENT*
- *POST KEY*
- *REDRAW*
- *SET ABOUT*
- *SET CURSOR*
- *SET FIELD TITLES*
- *SET TABLE TITLES*
- *Shift down*
- *SHOW MENU BAR*
- *Windows Alt down*
- *Windows Ctrl down*
- *_o_Get platform interface*
- *_o_INVERT BACKGROUND*
- *_o_SET PLATFORM INTERFACE*

BEEP

BEEP

Este comando no requiere parámetros

Descripción

*El comando **BEEP** hace que el PC o Macintosh generen un beep. Su ordenador (en Windows o Macintosh) puede emitir un sonido diferente a un beep, dependiendo del sonido seleccionado en el panel de control de sonido.*

***Advertencia:** no llame **BEEP** desde un proceso de conexión Web, porque el beep se producirá en el equipo servidor Web 4D y no el equipo del navegador Web.*

Ejemplo

En el siguiente ejemplo, si una búsqueda no encuentra ningún registro, se emite un beep y aparece un mensaje de alerta:

```
QUERY([Clientes];[Clientes]Nombre=$vsNombreaBuscar)
If(Records in selection([Clientes])=0)
   BEEP
   ALERT("No hay ningún cliente con ese nombre.")
End if
```

Caps lock down

Caps lock down -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*Estado de la tecla Bloq Mayús*

Descripción

Caps lock down *devuelve TRUE si la tecla Bloq Mayús está presionada.*

Ejemplo

Ver el ejemplo del comando ***Shift down****.*

Focus object

Focus object -> Resultado

Parámetro	Tipo		Descripción
Resultado	Puntero	➲	Puntero al objeto que tiene el foco

Nota de compatibilidad

*Este comando sólo se conserva por razones de compatibilidad. A partir de la versión 12 de 4D, se recomienda utilizar el comando **OBJECT Get pointer**.*

Descripción

***Focus object** devuelve un puntero al objeto que tiene el foco en el formulario actual. Si ningún objeto tiene el foco, el comando devuelve **Nil**. Puede utilizar **Focus object** para realizar una acción en un área de formulario sin saber cuál objeto está seleccionado actualmente. Asegúrese de probar si el objeto es del tipo correcto, utilizando **Type**, antes de realizar una acción.*

***Nota:** cuando se utiliza con un list box de tipo array, la función **Focus object** devuelve un puntero al list box o a la columna del list box que tiene el foco. En el caso de los list boxes de tipo selección, la función devuelve:*

- *para una columna asociada a un campo, un puntero al campo asociado,*
- *para una columna asociada a una variable, un puntero a la variable,*
- *para una columna asociada a una expresión, un puntero a la variable del list box.*

Este comando no puede utilizarse con campos en subformularios.

***Nota:** este comando se utiliza únicamente en el contexto de entrada de datos, de lo contrario se produce un error.*

Ejemplo

El siguiente ejemplo es un método de objeto para un botón. El método de objeto cambia los datos en el objeto actual a mayúsculas. El objeto debe ser del tipo texto o alfa (tipo 0 ó 24):

```
$vp :=Focus object  ` Obtener el puntero al último objeto
Case of
   :(Nil($puntero))  ` Ningún objeto tiene el foco
      ...
   :((Type($vp->)=ls alpha field)|(Type($vp->)=ls string var))  ` Si es un objeto de tipo texto o alfa
      $vp->:=Uppercase($vp->)  ` Cambiar los datos a mayúsculas
End case
```

GET FIELD TITLES

GET FIELD TITLES (tabla ; titulosCampos ; numCampos)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual quiere conocer los nombres de los campos
titulosCampos	Array texto	⬅	Nombres actuales de los campos
numCampos	Array entero largo	⬅	Números de los campos

Descripción

*El comando **GET FIELD TITLES** llena los arrays titulosCampos y numCampos con los nombres y los números de los campos de la tabla. Los contenidos de estos dos arrays están sincronizados.*

*Si el comando **SET FIELD TITLES** se llama durante la sesión, **GET FIELD TITLES** retorna únicamente los nombres "modificados" y los números de los campos definidos utilizando este comando.*

*De lo contrario, **GET FIELD TITLES** devuelve los nombres de los campos de la base definidos en la ventana de Estructura.*

En ambos casos, el comando no devuelve campos invisibles.

GET MOUSE

GET MOUSE (ratonX ; ratonY ; botonRaton {; *})

Parámetro	Tipo		Descripción
ratonX	Real	⇐	Coordenada horizontal del ratón
ratonY	Real	⇐	Coordenada vertical del ratón
botonRaton	Entero largo	⇐	Estado del botón del ratón: 0 = Botón arriba 1 = Botón presionado 2 = Botón derecho presionado 3 = Los dos botones presionados
*	Operador	⇒	Si se especifica, utlizar el sistema de coordenadas globales Si se omite, utilizar el sistema de coordenadas locales

Descripción

*El comando **GET MOUSE** devuelve el estado actual del ratón.*

*Las coordenadas horizontal y vertical se devuelven en ratonX y ratonY. Si pasa el parámetro *, las coordenadas se expresan con relación a la pantalla principal (modo macOS y Windows SDI) o a la ventana de la aplicación (modo Windows MDI). Si omite el parámetro *, se expresan con relación a la ventana del formulario actual(si la hay) del proceso actual .*

El parámetro botonRaton devuelve el estado de los botones, como se describió anteriormente.

***Nota:** los valores 2 y 3 pueden devolverse bajo Mac OS X a partir de la versión 10.2.5 únicamente.*

Ejemplo

*Ver el ejemplo del comando **Pop up menu**.*

GET TABLE TITLES

GET TABLE TITLES (titTablas ; numTablas)

Parámetro	Tipo		Descripción
titTablas	Array texto	⇐	Nombres actuales de las tablas
numTablas	Array entero largo	⇐	Números de las tablas

Descripción

El comando ***GET TABLE TITLES*** *llena los arrays titTablas y numTablas con los nombres y números de las tablas de la base definidas en la ventana de estructura o vía el comando* ***SET TABLE TITLES****. El contenido de estos dos arrays está sincronizado.*
Si el comando ***SET TABLE TITLES*** *se llama durante la sesión,* ***GET TABLE TITLES*** *sólo devuelve los nombres "modificados" y los números de las tablas definidos utilizando este comando.*

De lo contrario, ***GET TABLE TITLES*** *devuelve los nombres de las tablas de la base definidos en la ventana de estructura.*
En ambos casos, el comando no devuelve las tablas invisibles.

HIDE MENU BAR

HIDE MENU BAR
Este comando no requiere parámetros

Descripción

*El comando **HIDE MENU BAR** vuelve invisible la barra de menús.*
Si la barra de menús ya está oculta, el comando no hace nada.

Ejemplo

El siguiente método muestra un registro en toda la pantalla (Macintosh) hasta que haga clic:

```
HIDE TOOL BAR
HIDE MENU BAR
Open window(-1;-1;1+Screen width;1+Screen height;Alternate dialog box)
FORM SET INPUT([Pinturas];"Full Screen 800")
DISPLAY RECORD([Pinturas])
Repeat
   GET MOUSE($vlX;$vlY;$vlBoton)
Until($vlBoton#0)
CLOSE WINDOW
SHOW MENU BAR
SHOW TOOL BAR
```

***Nota:** en Windows, la ventana estará limitada por los límites de la ventana de la aplicación.*

Macintosh command down

Macintosh command down -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*Estado de la tecla Comando Macintosh o Estado de la tecla Ctrl en Windows*

Descripción

Macintosh command down *devuelve TRUE si la tecla Comando Macintosh está presionada.*
Nota: *cuando se llama bajo Windows,* ***Macintosh command down*** *devuelve TRUE si la tecla Ctrl Windows está presionada.*

Ejemplo

Ver el ejemplo del comando ***Shift down****.*

Macintosh control down

Macintosh control down -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano	⮌	Estado de la tecla Control de Macintosh

Descripción

Macintosh control down devuelve TRUE si la tecla Control de Macintosh está presionada.
Nota: cuando se llama bajo Windows, **Macintosh control down** devuelve FALSE. Esta tecla Macintosh no tiene equivalente en Windows.

Ejemplo

Ver el ejemplo del comando **Shift down**.

Macintosh option down

Macintosh option down -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*Estado de la tecla Opción Macintosh o Estado de la tecla Alt Windows*

Descripción

Macintosh option down *devuelve TRUE si la tecla Opción Macintosh está presionada.*
Nota: *cuando se llama bajo Windows,* ***Macintosh option down*** *devuelve TRUE si la tecla Alt Windows está presionada.*

Ejemplo

Ver el ejemplo del comando ***Shift down****.*

PLAY

PLAY (nomObjeto {; canal})

Parámetro	Tipo		Descripción
nomObjeto	Cadena	➡	Nombre de archivo de sonido o sonido sistema Cadena vacía para detener un sonido asincrónico
canal	Entero largo	➡	Si se pasa, canal de salida y ejecución asincrónico Si se omite, ejecución sincrónica

Descripción

*El comando **PLAY** permite reproducir archivos de sonido o multimedia. Pase la ruta de acceso completa del archivo que quiere reproducir en nombreObjeto. En OS X, el comando también puede utilizarse para reproducir un sonido sistema.*

- *Para reproducir un archivo, pase su nombre y ruta de acceso en nomObjeto. Puede pasar una ruta de acceso completa o relativa al archivo de estructura de la base.*
  Se soportan los principales formatos de archivos sonido y multimedia: .WAV, .MP3, .AVI, .AIFF (OS X), etc. Bajo OS X, el comando soporta particularmente formatos Core Audio.
- *(OS X únicamente) para reproducir un sonido sistema, pase directamente su nombre en el parámetro nomObjeto.*

***Nota**: los recursos 'snd', utilizados en Mac OS 9 y superiores, ya no se soportan.*

El parámetro asincrono permite reproducir de forma asíncrona en Windows. Síncrono significa que todo el procesamiento se detiene hasta que el sonido haya terminado de reproducirse, asíncrono significa que la reproducción no se detiene y el sonido se reproduce de fondo. Si se pasa asincrono y contiene 0 (o cualquier valor entero largo), el sonido se reproduce de forma asíncrona. Si se omite, el sonido se reproduce de forma sincrónica.

Para detener un sonido asincrónico, utilice la siguiente instrucción:

```
PLAY("";0)
```

Ejemplo 1

El siguiente ejemplo muestra cómo reproducir un archivo WAV en Windows:

```
$DocRef :=Open document("";"WAV";Read Mode)
If(OK=1)
   CLOSE DOCUMENT($DocRef)
   PLAY(Document;0) //reproducir asíncronamente
End if
```

Ejemplo 2

El siguiente código de ejemplo reproduce un sonido del sistema en OS X:

```
PLAY("Submarine.aiff")
```

Pop up menu

Pop up menu (contenido {; porDefecto {; CoordX ; CoordY}}) -> Resultado

Parámetro	Tipo		Descripción
contenido	Texto	➡	Definición del texto del menú
porDefecto	Entero largo	➡	Número del elemento seleccionado por defecto
CoordX	Entero largo	➡	Coordenada X de la esquina superior izquierda
CoordY	Entero largo	➡	Coordenada Y de la esquina superior izquierda
Resultado	Entero largo	⮌	Número de elemento de menú seleccionado

Descripción

*El comando **Pop up menu** muestra un menú pop up en la ubicación actual del ratón o en la ubicación definida por los parámetros opcionales coordX y coordY.*

Para seguir las reglas de interfaz de usuario, por lo general este comando debe llamarse en respuesta a un clic y si el botón del ratón aún está presionado.

Los elementos del menú pop up se definen con el parámetro contenido, de la siguiente manera:

- *Cada elemento se separa de los otros por un punto y coma (;). Por ejemplo, "Element1;Element2;Element3".*
- *Para desactivar un elemento, coloque un paréntesis abierto (() en el texto del elemento.*
- *Para definir una línea de separación, pase "-" o "(-" como texto del elemento.*
- *Para definir el estilo de fuente para una línea, coloque en el texto del elemento un signo menor que (<) seguido por uno de estos caracteres:*

<B Negrita
<I Itálica
<U Subrayado
<O Contorno (Macintosh únicamente)
<S Sombra (Macintosh únicamente)

- *Para añadir una marca de selección a un elemento, coloque en el texto del elemento un signo de admiración (!) seguido por el carácter que quiere utilizar como marca de selección.*

*- En Macintosh, el carácter se muestra directamente. Para mostrar la marca estándar sin importar la versión o el lenguaje del sistema, utilice la instrucción: **Char(18)**.*
- En Windows, se muestra una marca estándar, sin importar el carácter que pase.

- *Para añadir un icono a un elemento, coloque en el texto del elemento un acento circunflejo (^) seguido por un carácter cuyo código más 208 es el recurso del icono Mac OS.*
- *Para añadir un atajo a un elemento, coloque en el texto del elemento una barra oblicua (/) seguida por el carácter del atajo. Note que esta última opción es informativa únicamente; ningún atajo de teclado activa el menú pop up. Sin embargo, puede incluir un atajo si el elemento de menú pop up tiene un equivalente en la barra de menús principal de su aplicación.*

***Consejo:** es posible desactivar el mecanismo de interpretación de los caracteres especiales (!, /, etc.) en el menú pop up para, por ejemplo, tener estos caracteres incluidos en los textos. Para hacer esto, simplemente inicie el parámetro contenido con la instrucción **Char(1)** luego utilice esta instrucción como separador:*

```
contenido:=Char(1)+"1/4"+Char(1)+"1/2"+Char(1)+"3/4")
```

Note que una vez ejecutada esta instrucción, no es posible asignar estilos o atajos al menú pop up.

El parámetro opcional porDefecto le permite especificar el elemento de menú seleccionado por defecto cuando se muestra el menú. Pase un valor entre 1 y el número de elementos del menú. Si omite este parámetro, el comando selecciona por defecto el primer elemento del menú.

Los parámetros opcionales coordX y coordY se utilizan para designar la ubicación del menú pop-up a mostrar. En coordX y coordY, pase respectivamente las coordenadas horizontal y vertical de la esquina superior izquierda del menú. Estas coordenadas deben expresarse en píxeles en el sistema de coordenadas local del formulario actual. Estos dos parámetros deben pasarse juntos; si sólo se pasa uno, se ignorará.

Si utiliza los parámetros coordX y coordY, el parámetro por defecto se ignora. En este caso, el ratón no se encuentra necesariamente en el nivel del menú pop up.

Estos parámetros son útiles en particular para administrar los botones 3D con un menú pop up asociado.

Si selecciona un elemento de menú, el comando devuelve su número; de lo contrario, devuelve cero (0).

***Nota:** utilice los menús pop up con un número razonable de elementos. Si quiere mostrar más de 50 elementos, puede utilizar mejor un área de desplazamiento en un formulario.*

Ejemplo

*El método de proyecto **MI MENU RAPIDO** hace aparecer un menú de navegación pop up:*

```
` Método de proyecto MI MENU RAPIDO
GET MOUSE($vlRatonX;$vlRatonY;$vlBoton)
If(Macintosh control down|($vlBoton=2))
   $vtElementos:="Sobre esta base...<l;(-;!-Otras opciones;(-"
```

```
    For($vlTabla;1;Get last table number)
        If(Is table number valid($vlTabla))
            $vtElementos:=$vtElementos+";"+Table name($vlTabla)
        End if
    End for
    $vlEleccionUsuario:=Pop up menu($vtElementos)
    Case of
        :($vlEleccionUsuario=1)
    ` Mostrar información
        :($vlEleccionUsuario=2)
    ` Mostrar las opciones
        Else
            If($vlEleccionUsuario>0)
    ` Ir a la tabla cuyo número es $vlEleccionUsuario-4
            End if
    End case
End if
```

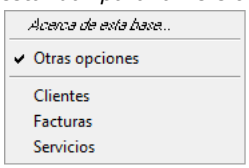
Este método de proyecto puede llamarse desde:

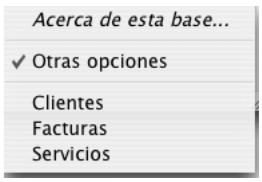
- *el método de un objeto de formulario que reacciona a un clic sin esperar a que el botón del ratón sea liberado (por ejemplo un botón invisible)*
- *un proceso que "espía" los eventos y se comunica con los otros procesos*
- *un método de gestión de eventos instalado por el comando **ON EVENT CALL**.*

*En los dos últimos casos, no es necesario que el clic se produzca en un objeto de formulario. Esta es una de las ventajas del comando **Pop up menu**. Generalmente, los menús pop up se muestran por medio de objetos de formulario. Con **Pop up menu**, puede mostrar un menú pop up en cualquier lugar.*

El menú pop up se muestra en Windows al presionar el botón derecho del ratón, y en Macintosh al presionar Control-Clic. Note, sin embargo, que el método no verifica si en realidad hubo un clic; el método llamante hace esa prueba.

El siguiente es el menú pop-up tal como aparece en Windows (izquierda) y Macintosh (derecha). Note la marca de selección estándar para la versión Windows.





POST CLICK

POST CLICK (ratonX ; ratonY {; proceso} {; *})

Parámetro	*Tipo*		*Descripción*
ratonX	*Entero largo*	➡	*Coordenada horizontal*
ratonY	*Entero largo*	➡	*Coordenada vertical*
proceso	*Entero largo*	➡	*Número de referencia del proceso de destino o fila de atención de eventos de la aplicación, si se omite o si se pasa 0*
*		➡	*Si se especifica, utilizar el sistema de coordenadas globales Si se omite, utilizar el sistema de coordenadas locales*

Descripción

*El comando **POST CLICK** simula un clic del ratón. Tiene el mismo efecto que cuando el usuario hace clic en el botón del ratón.*

*Pase las coordenadas horizontal y vertical del clic en ratonX y ratonY. Si pasa el parámetro *, expresa estas coordenadas con respecto a la pantalla. Si omite el parámetro *, expresa estas coordenadas con relación a la ventana del primer plano del proceso cuyo número de proceso se pasa en proceso.*

Si especifica el parámetro proceso proceso, el clic se envía al proceso cuyo número de proceso se pasa en proceso. Si pasa 0 (cero) o si omite el parámetro, el clic se envía a nivel de la aplicación y el planificador 4D la enviará al proceso apropiado.

POST EVENT

POST EVENT (que ; mensaje ; cuando ; ratonX ; ratonY ; modificadores {; proceso})

Parámetro	Tipo		Descripción
que	Entero largo	➡	Tipo de evento
mensaje	Entero largo	➡	Mensaje del evento
cuando	Entero largo	➡	Momento del evento expresado en tics
ratonX	Entero largo	➡	Coordenada horizontal del ratón
ratonY	Entero largo	➡	Coordenada vertical del ratón
modificadores	Entero largo	➡	Estado de las teclas Modificadores
proceso	Entero largo	➡	Número de referencia del proceso de destino o Fila de atención de los eventos de la aplicación, si se omite o si se pasa 0

Descripción

*El comando **POST EVENT** simula un evento de teclado o de ratón. Tiene el mismo efecto que cuando el usuario actúa a través del teclado o del ratón.*

Pase una de las siguientes constantes predefinidas en que:

Constante	Tipo	Valor
Mouse down event	Entero largo	1
Mouse up event	Entero largo	2
Key down event	Entero largo	3
Key up event	Entero largo	4
Auto key event	Entero largo	5

Si el evento es un evento relacionado con el ratón, pase 0 (cero) en mensaje. Si el evento es un evento relacionado con el teclado, pase el código del carácter simulado en mensaje.

*Generalmente, se pasa el valor devuelto por **Tickcount** en cuando.*

Si el evento es un evento relacionado con el ratón, pase las coordenadas horizontal y vertical del clic en ratonX y ratonY.

*En el parámetro modificadores, pase una constante o una combinación de constantes del tema . Por ejemplo, para simular la tecla Mayús, pase **Shift key bit**.*

Si pasa el parámetro proceso, el evento se envía al proceso cuyo número se pasa en proceso. Si pasa 0 (cero) o si omite este parámetro, el evento se envía a nivel de la aplicación y el planificador de 4D lo enviará al proceso apropiado.

POST KEY

POST KEY (codigo {; modificadores {; proceso}})

Parámetro	*Tipo*		*Descripción*
codigo	*Entero largo*	➡	*Código de un caracter o código de tecla de función*
modificadores	*Entero largo*	➡	*Estado de teclas Modificador*
proceso	*Entero largo*	➡	*Número de referencia del proceso de destino o Fila de atención de eventos de la aplicación, si se omite o si es igual a 0*

Descripción

*El comando **POST KEY** simula una tecla. Tiene el mismo efecto que cuando un usuario digita un carácter en el teclado.*

Pase el código del carácter en codigo.

*Si pasa el parámetro modificadores, pase una constante o una combinación del constantes del tema Events (modifiers). Por ejemplo, para simular la tecla Mayús, pase **Shift key mask**.*

Si pasa el parámetro proceso, la tecla se envía al proceso cuyo número de referencia se especifica en proceso. Si pasa 0 (cero) o si omite el parámetro, la tecla se envía al nivel de la aplicación y el planificador de 4D la enviará al proceso apropiado.

Ejemplo

*Ver el ejemplo del comando **Process number**.*

REDRAW

REDRAW (objeto)

Parámetro	Tipo		Descripción
objeto	Objeto de formulario	➡	Tabla para la cual rediseñar el subformulario o Campo para el cual rediseñar el área o Variable para la cual rediseñar el área o Tabla del formulario a rediseñar en un navegador Web

Descripción

Cuando utiliza un método para modificar el contenido de un campo o subcampo mostrado en un subformulario, debe ejecutar ***REDRAW*** *para asegurar que el formulario esté correctamente actualizado. .*

En el contexto de los list boxes en modo selección, la instrucción ***REDRAW*** *aplicada a un objeto de tipo list box provoca la actualización de los datos mostrados en el objeto. Esta instrucción debe llamarse típicamente después de una modificación de los datos en los registros de la selección.*

SET ABOUT

SET ABOUT (textoElem ; metodo)

Parámetro	Tipo		Descripción
textoElem	Cadena	➡	Nueva línea de menú Acerca de
metodo	Cadena	➡	Nombre del método a ejecutar cuando se elije la línea

Descripción

*El comando **SET ABOUT** cambia la línea de menú Acerca de 4D del menú **Ayuda** (Windows) o del menú **Aplicación** (Mac OS X) por textoLinea.*

Después de llamar este comando, cuando el usuario selecciona esta línea de menú en el entorno Diseño o Aplicación, se llama metodo. Generalmente, este método muestra una caja de diálogo que da información sobre la versión de su base.

Este comando se utiliza con 4D (local y remoto), 4D Desktop y 4D Server. Su ejecución en el equipo servidor provoca la creación de un nuevo proceso.

Ejemplo 1

*El siguiente ejemplo reemplaza el comando de menú Acerca de 4D por el comando de menú Acerca del programador. El método **ACERCA DE** muestra una caja Acerca de personalizada:*

```
SET ABOUT(Acerca del programador;ACERCA DE)
```

Ejemplo 2

El siguiente ejemplo reinicializa el comando de menú Acerca de 4D:

```
SET ABOUT("Acerca de 4D";"")
```

SET CURSOR

SET CURSOR {(cursor)}

Parámetro	Tipo		Descripción
cursor	Entero largo	➡	Número de cursor sistema

Descripción

*El comando **SET CURSOR** cambia el puntero (gráfico) del ratón por el del sistema cuyo número de identificación se pasa en cursor.*
*Este comando debe llamarse en el contexto del **Evento formulario** On Mouse Move.*
Para restablecer el cursor de ratón estándar, llame el comando sin parámetro.
Los siguientes son algunos cursores que pueden pasarse en el parámetro cursor:

Number	Cursor
1	
2	
4	
9000	
9001	
9003	
9004	
9005	
9006	
9021	
351	
9010	
9011	
9013	
9014	
9015	
9016	
9017	
9019	
9020	
559	
560	

Nota: *la disponibilidad y la apariencia de los cursores puede variar dependiendo del sistema operativo.*

Ejemplo

Usted quiere que se muestre el cursor 🚫 cuando el ratón pase sobre una área variable en el formulario. Puede escribir en el método objeto de la variable:

```
If(Form event=On Mouse Move)
   SET CURSOR(9019)
End if
```

SET FIELD TITLES

SET FIELD TITLES (tabla ; titulosCampos ; numCampos {; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla o Subtabla para la cual definir los títulos de los campos
titulosCampos	Array cadena	➡	Nuevos títulos de los campos
numCampos	Array entero largo	➡	Números de los campos
*		➡	Utilizar los nombres personalizados en el editor de fórmulas

Descripción

SET FIELD TITLES *permite ocultar, renombrar y reorganizar los campos de una tabla o de una subtabla pasada en* tabla *o* subtabla *cuando aparecen en los editores estándar de 4D, tal como el editor de búsquedas, en modo Aplicación (más específicamente, cuando los editores son llamados vía los comandos del lenguaje de 4D).*

Utilizando este comando, puede también renombrar instantáneamente las etiquetas de los campos en sus formularios, si ha utilizado nombres dinámicos. Para mayor información sobre la inserción de etiquetas de tablas y campos dinámicos en los formularios, consulte el Manual de Diseño 4D.

Los arrays titulosCampos *y* numCampos *deben estar sincronizados. En el array* titulosCampos*, pase el nombre de los campos tal como quiere que aparezcan. Si no quiere mostrar un campo en particular, no incluya su nombre o nuevo título en el array. Los campos aparecerán en el orden que especifique en este array. En cada elemento del array* numCampos*, pase el número de la tabla que corresponde al nombre, nuevo o antiguo, del campo pasado en el mismo número de elemento que en el array* titulosCampos*.*

Por ejemplo, usted tiene una tabla o subtabla compuesta por los campos F, G, y H, creada en ese orden. Usted quiere que estos campos aparezcan como M, N, y O. Además usted no quiere mostrar el campo N. Por último, quiere mostrar O y M en ese orden. Para hacer esto, pase en el parámetro titulosCampos *un array que contenga dos elementos, O y M, y pase en el parámetro* numCampos *un array que contenga dos elementos, 3 y 1.*

*El parámetro opcional * indica si los nombres personalizados (estructuras "virtuales") definidos utilizando este comando puede ser utilizados o no en fórmulas 4D.*

- *Por defecto, cuando se omite este parámetro, las fórmulas ejecutadas en 4D no pueden utilizar estos nombres personalizados; es necesario utilizar los nombres de campos reales. Este principio da una cierta libertad para dar nombres a los campos ya que el intérprete del lenguaje no procesa nombres personalizados.*
- *Si se pasa el parámetro *, los nombres definidos por este comando pueden utilizarse en las fórmulas ejecutadas por 4D.* ***Tenga cuidado en este caso****, los nombres personalizados no deben contener caracteres que son considerados como "prohibidos" por el interprete del lenguaje 4D, tal como -?*! (para mayor información, consulte la sección "****Convenciones****").*

Nota: *si su aplicación da acceso al editor de fórmulas (por ejemplo por el editor de Informes rápidos), es necesario pasar el parámetro * para mantener consistencia en la aplicación.*

SET FIELD TITLES *NO modifica la estructura de su base. Sólo afecta la visualización posterior de los editores estándar de 4D y de los formularios que utilizan nombres dinámicos cuando se llaman vía un comando del lenguaje (la estructura real de la base se muestra cuando el editor o formulario se llama desde un comando de menú en modo Diseño). El alcance del comando* ***SET FIELD TITLES*** *es la sesión de trabajo. Un beneficio en cliente/Servidor es que varias estaciones 4D cliente pueden "ver" simultáneamente su estructura de una manera diferente. Puede llamar* ***SET FIELD TITLES*** *tantas veces como quiera.*

Utilice el comando ***SET FIELD TITLES*** *para:*

- *Traducir dinámicamente su base.*
- *Mostrar los campos en el orden que usted quiera, independientemente de la definición real de su tabla.*
- *Mostrar los campos de manera que dependan de la identidad o de los privilegios personalizados de un usuario.*

Advertencia:

- ***SET FIELD TITLES*** *NO anula el efecto de la propiedad invisible de un campo. Cuando un campo se define para que sea invisible al nivel de la estructura, no aparecerá en modo Aplicación, aunque se incluya en una llamada a* ***SET FIELD TITLES****.*
- *Los plug-ins siempre acceden a la estructura "virtual" como está especificado por este comando.*
- *Si se llama el comando* ***SET TABLE TITLES*** *sin parámetros, el entorno del lenguaje se reinicia y la estructura "virtual" (incluyendo los nombres de tablas y campos personalizados) se elimina completamente.*

Ejemplo

Ver el ejemplo del comando ***SET TABLE TITLES****.*

SET TABLE TITLES

SET TABLE TITLES {(titulosTablas ; numTablas {; *})}

Parámetro	Tipo		Descripción
titulosTablas	Array cadena	➡	Nombres de las tablas tal como deben aparecer
numTablas	Array entero largo	➡	Números de las tablas
*		➡	Utilizar los nombres personalizados en el editor de fórmulas

Descripción

SET TABLE TITLES *permite ocultar, renombrar y reordenar las tablas de su base cuando aparecen en los editores estándar de 4D en modo Aplicación (cuando los editores se llaman vía los comandos del lenguaje 4D).Por ejemplo, este comando puede modificar la visualización de tablas en el editor búsquedas en modo Aplicación.*

Utilizando este comando, también puede renombrar instantáneamente las etiquetas de las tablas en sus formularios, si utilizó nombres dinámicos. Para mayor información sobre la inserción de etiquetas de tablas y de campos dinámicos en formularios, consulte la sección ***Utilizar las referencias en los textos estáticos*** *en el manual de Diseño de 4D.*

Los arrays titulosTablas y numTablas deben estar sincronizados. En el array titulosTablas, pase los nombres de las tablas como quiere que aparezcan. Si no quiere mostrar una tabla en particular, no incluya su nombre o nuevo título en el array. Las tablas aparecerán en el orden que especificó en este array. En cada elemento del array numTablas, pase el número de la tabla que corresponde al nombre, nuevo o antiguo, de la tabla, pasado en el mismo número de elemento en el array titulosTablas.

Por ejemplo, usted tiene una base compuesta por las tablas A, B, y C, creadas en este orden. Usted quiere que estas tablas aparezcan como X, Y, y Z. Además no quiere mostrar la tabla B. Por último, quiere mostrar Z y X, en este orden. Para hacer esto, pase en el parámetro titulosTablas un array de dos elementos, Z y X, y pase en el parámetro numTablas un array de dos elementos, 3 y 1.

*El parámetro opcional * le permite indicar si los nombres personalizados (estructuras"virtuales") definidos utilizando este comando son utilizados o no en las fórmulas de 4D:*

- *Por defecto, cuando se omite este parámetro, las fórmulas ejecutadas en 4D no pueden utilizar estos nombres personalizados; es necesario utilizar los nombres verdaderos de las tablas.*
  Utilizar nombres personalizados da mayor libertad para dar nombres a las tablas ya que el intérprete del lenguaje no procesa nombres personalizados.
- *Si se pasa el parámetro *, los nombres definidos por este comando pueden utilizarse en las fórmulas ejecutadas por 4D.* ***Atención en este caso****, los nombres personalizados no deben contener caracteres que estén "prohibidos" por el interprete del lenguaje de 4D, tal como -?*! Por ejemplo, el nombre "Rate_in_%" no puede utilizarse en una fórmula. (para mayor información, consulte la sección* ***Convenciones****).*

Nota: *si su aplicación da acceso al editor de fórmulas (por ejemplo por el editor de Informes rápidos), es necesario pasar el parámetro * para mantener consistencia en la aplicación.*

SET TABLE TITLES *NO cambia la estructura actual de su base. Sólo afecta los usos posteriores de los editores estándar de 4D y formularios que utilizan nombres dinámicos cuando se llaman vía un comando del lenguaje (la estructura real de la base se muestra cuando el editor o formulario se llama desde un comando de menú en el entorno Diseño). El alcance del comando* ***SET TABLE TITLES*** *es la sesión de trabajo. La ventaja, en cliente/Servidor es que varios puestos clientes 4D pueden "ver" simultáneamente su base de diferentes formas. Puede llamar* ***SET TABLE TITLES*** *tantas veces como lo considere necesario.*

Utilice el comando ***SET TABLE TITLES*** *para:*

- *Traducción dinámica de su base.*
- *Visualización de las tablas en el orden que quiera, independientemente del la definición actual de su base.*
- *Visualización de tablas que dependa de la identidad o de los privilegios de un usuario.*

ATENCIÓN:

- ***SET TABLE TITLES*** *NO anula el efecto de la propiedad invisible de una tabla. Cuando una tabla está definida como invisible a nivel de la estructura de su base, aunque se incluya en una llamada a* ***SET TABLE TITLES****, no aparecerá en modo Aplicación.*
- *Los plug-ins siempre acceden a la estructura "virtual" como se especifica por este comando.*

Ejecutar ***SET TABLE TITLES*** *sin parámetros reiniciará toda la estructura virtual de la base para la sesión (tabla personalizada y nombres de campos).*

Ejemplo 1

- *Usted desarrolla una aplicación 4D que piensa vender internacionalmente. Por lo tanto, usted debe tener en cuenta las necesidades de traducción. Para los editores estándar de 4D que aparecen en modo Aplicación y sus formularios que utilizan nombres dinámicos, puede utilizar una tabla [Traducciones] y algunos métodos de proyecto para crear y utilizar las traducciones para cada idioma.*

- *En su base, cree la siguiente tabla:*

- *Luego, cree el método de proyecto TRADUCIR_TABLAS_Y_CAMPOS. Este método analiza la estructura de su base en la tabla [Traducciones] y crea los registros correspondientes al idioma pasado como parámetro.*

```
  ` Método de proyecto TRADUCIR_TABLAS_Y_CAMPOS
  ` TRADUCIR_TABLAS_Y_CAMPOS (Text)
  ` TRADUCIR_TABLAS_Y_CAMPOS (LanguageCode)

C_TEXT($1) `código del idioma
C_LONGINT($vlTabla;$vlCampo)
C_TEXT($Idioma)
$Idioma:=$1

For($vlTabla;1;Get last table number) `Pasar por cada tabla
   If($vlTabla#(Table(->[Traducciones]))) `No traducir la tabla de traducciones
  `Verificar si existe una traducción de nombre de la tabla para el idioma especificado
      QUERY([Traducciones];[Traducciones]Codigo_Idioma=$Idioma;*) `idioma deseado
      QUERY([Traducciones];&;[Traducciones]Tabla_ID=$vlTabla;*) `Número de tabla
      QUERY([Traducciones];&;[Traducciones]Campo_ID=0) `número de campo = 0 significa que es un nombre de tabla
      If(Is table number valid($vlTabla)) `verificar que la tabla aún existe
         If(Records in selection([Traducciones])=0)
  `De lo contrario, crear el registro
            CREATE RECORD([Traducciones])
            [Traducciones]Codigo_Idioma:=$Idioma
            [Traducciones]Tabla_ID:=$vlTabla
            [Traducciones]Campo_ID:=0
  `El nombre de la tabla traducida deberá introducirse
            [Traducciones]Traduccion:=Table name($vlTabla)+" en "+$Idioma
            SAVE RECORD([Traducciones])
         End if

         For($vlCampo;1;Obtener número del último campo($vlTabla))
  `Verificar si exise una traducción para el nombre del campo en el idioma especificado
            QUERY([Traducciones];[Traducciones]Codigo_Idioma=$Idioma;*) `idioma deseado
            QUERY([Traducciones];&;[Traducciones]Tabla_ID=$vlTabla;*) `número de tabla
            QUERY([Traducciones];&;[Traducciones]Campo_ID=$vlCampo) `número de campo
            If(Is field number valid($vlTabla;$vlCampo))
               If(Records in selection([Traducciones])=0)
  `De lo contrario, crear el registro
                  CREATE RECORD([Traducciones])
                  [Traducciones]Codigo_Idioma:=$Idioma
                  [Traducciones]Tabla_ID:=$vlTabla
                  [Traducciones]Campo_ID:=$vlCampo
  `El nombre del campo traducido debe introducirse
                  [Traducciones]Traduccion:=Field name($vlTabla;$vlCampo)+"en "+$Idioma
                  SAVE RECORD([Traducciones])
               End if
            Else
               If(Records in selection([Traducciones])#0)
  ` Si el campo no existe, eliminar la traducción
                  DELETE RECORD([Traducciones])
               End if
            End if
         End for
      Else
         If(Records in selection([Traducciones])#0)
  `Si la tabla no existe, eliminar la traducción
            DELETE RECORD([Traducciones])
         End if
      End if
   End if
End for
```

- *En este punto, si ejecuta la siguiente línea, puede crear tantos registros como necesite para la traducción al Español de sus tablas y campos.*

```
TRANSLATE TABLES AND FIELDS("Español")
```

- *Una vez ejecutada esta llamada, puede introducir una traducción en el campo [Traducciones]Nombre traducido para cada uno de los nuevos registros.*
- *Por último, cada vez que quiera mostrar en español los editores estándar 4D o los formularios con etiquetas dinámicas, ejecute la siguiente línea:*

```
LOCALIZED TABLES AND FIELDS("Español")
```

*con el método de proyecto **TABLAS_Y_CAMPOS_LOCALIZADOS**:*

```
  `Método objeto TABLAS_Y_CAMPOS_LOCALIZADOS
  ` TABLAS_Y_CAMPOS_LOCALIZADOS (Text)
  ` TABLAS_Y_CAMPOS_LOCALIZADOS (LanguageCode)

C_TEXT($1) `Código del idioma
C_LONGINT($vlTabla;$vlCampo)
C_TEXT($Idioma)
C_LONGINT($vlNumTabla;$vlNumCampo)
$Idioma:=$1

  `Actualización de los nombres de tablas
ARRAY TEXT($asNombres;0) `Inicializar los arrays para SET TABLE TITLES y SET FIELD TITLES
ARRAY INTEGER($aiNumeros;0)
QUERY([Traducciones];[Traducciones]Codigo_Idioma=$Idioma;*)
QUERY([Traducciones];&;[Traducciones]Campo_ID=0) `nombres de tablas
SELECTION TO ARRAY([Traducciones]Traduccion;$asNombres;[Traducciones]Tabla_ID;$aiNumeros)
SET TABLE TITLES($asNombres;$aiNumeros)

  `Actualización de los nombres de campos
$vlNumTabla:=Get last table number `Obtener el número de tablas en la base
For($vlTabla;1;$vlNumTabla) `Pasar por cada tabla
   If(Is table number valid($vlTabla))
      QUERY([Traducciones];[Traducciones]Codigo_Idioma=$Idioma;*)
      QUERY([Traducciones]; & ;[Traducciones]Tabla_ID=$vlTabla;*)
      QUERY([Traducciones]; & ;[Traducciones]Campo_ID#0) `evite que el cero sea nombre de tabla
      SELECTION TO ARRAY([Traducciones]Traduccion;$asNombres;[Traducciones]Campo_ID;$aiNumeros)
      SET FIELD TITLES(Tabla($vlTabla)->;$asNombres;$aiNumeros)
   End if
End for
```

- *Note que las nuevas localizaciones pueden añadirse a la base sin modificar el código o recompilarlo.*

Ejemplo 2

Desea eliminar todos los nombres de tablas y campos personalizados definidos:

```
SET TABLE TITLES //elimina todos los nombres personalizados
```

Shift down

Shift down -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*Estado de la tecla Mayús*

Descripción

Shift down *devuelve TRUE si la tecla Mayús está presionada.*

Ejemplo

El siguiente método de objeto para el botón bUnBoton efectúa diferentes acciones en función de las teclas de modificación presionadas cuando se hace clic en el botón:

```
  ` Método de objeto bUnBoton
Case of
  ` Otras combinaciones diferentes pueden probarse aquí
  ` ...
   :(Shift down&Windows Ctrl down)
  ` Las teclas Mayús y Ctrl Windows (o Comando Mac OS) son presionadas
      DO ACTION1
  ` ...
   :(Shift down)
  ` Sólo está presionada Mayús
      DO ACTION2
  ` ...
   :(Windows Ctrl down)
  ` Sólo está presionada Ctrl Windows (o Comando Mac OS)
      DO ACTION3
  ` ...
  ` Otras teclas pueden probarse individualmente aquí
  ` ...
End case
```

SHOW MENU BAR

SHOW MENU BAR
Este comando no requiere parámetros

Descripción

*El comando **SHOW MENU BAR** hace visible la barra de menús.*
Si la barra de menús ya era visible, el comando no hace nada.

Ejemplo

*Ver el ejemplo del comando **HIDE MENU BAR**.*

Windows Alt down

Windows Alt down -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano	➲	Estado de la tecla Windows Alt Estado de la tecla Macintosh Opción

Descripción

Windows Alt down *devuelve TRUE si la tecla Alt Windows está presionada.*
Nota: *cuando se llama en una plataforma Macintosh,* ***Windows Alt down*** *devuelve TRUE si la tecla Macintosh Opción está presionada.*

Ejemplo

Ver el ejemplo del comando ***Shift down****.*

Windows Ctrl down

Windows Ctrl down -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*Estado de la tecla Ctrl Windows o Estado de la tecla Comando Macintosh*

Descripción

Windows Ctrl down *devuelve TRUE si la tecla Ctrl Windows está presionada.*
Nota: *cuando se llama en una plataforma Macintosh,* ***Windows Ctrl down*** *devuelve TRUE si la tecla Macintosh Comando está presionada.*

Ejemplo

Ver el ejemplo del comando ***Shift down****.*

_o_Get platform interface

_o_Get platform interface -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Interfaz de plataforma en uso

Nota de compatibilidad

La interfaz de plataforma es generada automáticamente por 4D. Este comando no debe utilizarse ya que ya no devuelve resultados significativos.

_o_INVERT BACKGROUND

_o_INVERT BACKGROUND ({* ;} textObjeto)

Parámetro	Tipo		Descripción
*	Operador	➡	Permite la entrada de una variable o nombre de objeto
textObjeto	Variable, Campo	➡	Variable o campo de tipo texto a invertir

Descripción

Este comando ya no es soportado en 4D.

_o_SET PLATFORM INTERFACE

_o_SET PLATFORM INTERFACE (interfaz)

Parámetro	Tipo		Descripción
interfaz	Entero largo	➡	Nueva interfaz de plataforma: -1 Plataforma automática 0 Mac OS 7 1 Windows 3.11, NT 3.51 2 Windows 9x 3 Mac OS 9 4 Mac Theme

Nota de compatibilidad

La interfaz de plataforma es manejada automáticamente por 4D. Este comando ahora se ignora y no debe ser utilizado.

Interrupciones

- *ABORT*
- *ASSERT*
- *Asserted*
- *FILTER EVENT*
- *Get assert enabled*
- *GET LAST ERROR STACK*
- *Method called on error*
- *Method called on event*
- *ON ERR CALL*
- *ON EVENT CALL*
- *SET ASSERT ENABLED*

ABORT

ABORT
Este comando no requiere parámetros

Descripción

*El comando **ABORT** se utiliza en un método de proyecto de gestión de errores instalado por el comando **ON ERR CALL**.*
Si no instala un método de proyecto de gestión de errores, cuando ocurra un error (por ejemplo, un error de la base de datos) 4D mostrará su caja de diálogo de error estándar y luego interrumpirá la ejecución de su código. Si el código en ejecución es:

- *Un método de objeto, método de formulario (o un método de proyecto llamado por un método de formulario o de objeto), el control vuelve al formulario que se muestra actualmente.*
- *Un método llamado desde un menú, el control vuelve a la barra de menús o formulario que se muestra actualmente.*
- *El método maestro de un proceso, entonces el proceso termina.*
- *Un método llamado directa o indirectamente por una operación de importación o exportación, la operación se detiene. Lo mismo para las búsquedas secuenciales u ordenaciones.*
- *Etc.*

Si utiliza un método de proyecto de intercepción de errores, 4D no muestra más su caja de diálogo de errores estándar y no interrumpe la ejecución de su código. En lugar de eso, 4D llama a su método de proyecto de intercepción de errores (que puede ver como un manejador de excepciones), y reasume la ejecución de la línea de código siguiente en el método que disparó el error.
*Hay errores que puede tratar por programación; por ejemplo, durante una operación de importación, si intercepta un error de la base de datos que señala un valor duplicado, puede "cubrir" el error y seguir con la importación. Sin embargo, hay errores que no puede procesar y errores que no debe "cubrir." En estos casos, necesita detener la ejecución llamando **ABORT** desde el método de proyecto de intercepción de errores.*

Nota histórica

*Aunque el comando **ABORT** está destinado a ser utilizado sólo desde un método de proyecto de intercepción de errores, algunos miembros de la comunidad 4D también lo utilizan en otros métodos para interrumpir su ejecución. El hecho de que funcione es sólo un efecto secundario. No recomendamos utilizar este comando en métodos diferentes a los métodos de proyecto de intercepción de errores.*

ASSERT

ASSERT (expresionBool {; textoMensaje})

Parámetro	Tipo		Descripción
expresionBool	Booleano	➡	Expresión booleana
textoMensaje	Texto	➡	Texto del mensaje de error

Descripción

*El comando **ASSERT**evalúa la aserción expresionBool pasada en parámetro y, si se vuelve falsa, interrumpe la ejecución del código y muestra un error.*

El comando funciona en modo interpretado y en modo compilado.

*Si la expresión es verdadera, no pasa nada. Si es falsa, el comando dispara el error -10518 y muestra por defecto el texto de la aserción precedido del mensaje "Aserción fallida:". Puede interceptar este error vía un método instalado utilizando el comando **ON ERR CALL**, para por ejemplo alimentar un archivo de historial.*

El comando acepta un segundo parámetro opcional que puede utilizarse para proporcionar un texto que se mostrará en el mensaje de error en lugar de la expresión booleana cuando sea falsa.

Opcionalmente, puede pasar un parámetro textoMensaje para mostrar un mensaje de error personalizado en lugar del texto de la aserción.

*Una aserción es un instrucción insertada en el código que es responsable de detectar posibles anomalías durante su ejecución. El principio consiste en verificar que una expresión es verdadera en un momento dado y en caso contrario, producir una excepción. Las aserciones se utilizan sobre todo para detectar casos que no deberían ocurrir nunca. Principalmente se utilizan para detectar bugs de programación. Es posible activar o desactivar globalmente todas las aserciones de una aplicación (por ejemplo de acuerdo al tipo de versión) vía el comando **SET ASSERT ENABLED**. Para más información acerca de las aserciones en programación, por favor consulte el artículo en Wikipedia: http://en.wikipedia.org/wiki/Assertion_(computing)*

Ejemplo 1

Antes de efectuar operaciones en un registro, el desarrollador quiere asegurarse de que está cargado en modo lectura/escritura:

```
READ WRITE([Tabla 1])
LOAD RECORD([Tabla 1])
ASSERT(Not(Locked([Tabla 1])))
  // dispara el error -10518 si el registro está bloqueado
```

Ejemplo 2

Una aserción permite probar los parámetros pasados a un método de proyecto para detectar los valores aberrantes. En este ejemplo, se utiliza un mensaje de alerta personalizado.

```
  // Método que devuelve el número de un cliente en función de su nombre pasado en $1
C_TEXT($1) // Nombre del cliente
ASSERT($1#"";"Búsqueda de un nombre de cliente vacío")
  // Un nombre vacío en este caso es un valor aberrante
  // Si la aserción es falsa, se mostrará en la caja de diálogo el error:
  // "Aserción fallida: búsqueda de un nombre de cliente vacía"
```

Asserted

Asserted (expresionBool {; textoMensaje}) -> Resultado

Parámetro	Tipo		Descripción
expresionBool	Booleano	➡	Expresión booleana
textoMensaje	Texto	➡	Texto del mensaje de error
Resultado	Booleano	⮌	Resultado de la evaluación de expresionBool

Descripción

*El comando **Asserted** tiene un funcionamiento similar al del comando **ASSERT**, con la diferencia de que devuelve un valor que es el resultado de la evaluación del parámetro expresionBool. Permite utilizar una aserción durante la evaluación de una condición (ver el ejemplo). Para mayor información sobre el funcionamiento de las aserciones y los parámetros de este comando, consulte la descripción del comando **ASSERT**.*

***Asserted** acepta una expresión Booleana como parámetro y devuelve el resultado de la evaluación de esta expresión. Si la expresión es falsa y si las aserciones están activas (ver el comando **SET ASSERT ENABLED**), se genera el error -10518, exactamente que para el comando **ASSERT**. Si las aserciones están inactivas, **Asserted** devuelve el resultado de la expresión que se pasó sin disparar un error.*

***Nota:** como el comando **ASSERT**, **Asserted** funciona en modo interpretado y en modo compilado.*

Ejemplo

Inserción de una aserción en la evaluación de una expresión:

```
READ WRITE([Tabla 1])
LOAD RECORD([Tabla 1])
If(Asserted(Not(Locked([Tabla 1]))))
  // Este código desencadena el error -10518 si el registro está bloqueado
    ...
End if
```

FILTER EVENT

FILTER EVENT
Este comando no requiere parámetros

Descripción

*El comando **FILTER EVENT** debe llamarse desde el interior del método de gestión de eventos instalado utilizando el comando **ON EVENT CALL**.*

*Sin un método de gestión de eventos llama a **FILTER EVENT**, el evento actual no pasa a 4D.*

Este comando le permite remover el evento actual (por ejemplo, clic, digitación de teclas) de la secuencia de eventos, de manera que 4D no efectúe ningún tratamiento adicional al que usted provocó en el método de gestión de eventos.

***Advertencia:** evite crear un método de gestión de eventos que sólo llame al comando **FILTER EVENT**, porque todos los eventos van a ser ignorados por 4D. En caso que tenga un método de gestión de eventos con el comando **FILTER EVENT** solamente, digite Ctrl+Mayús+Retroceso (en Windows) o comando-Opción-Mayús-Control-Retroceso (en Macintosh). Esto convierte el proceso On Event Call en un proceso normal que no obtiene eventos.*

***Caso especial:** el comando **FILTER EVENT** puede igualmente utilizarse en un método de formulario estándar cuando el formulario se visualiza utilizando los comandos **DISPLAY SELECTION** o **MODIFY SELECTION**. En este caso específico, el comando **FILTER EVENT** le permite filtrar los doble clics en los registros (y de esta manera ejecutar acciones diferentes a las de apertura de los registros en modo página).*
Para hacer esto, coloque las siguientes líneas en el método del formulario de salida:

```
If(Form event=On Double Clicked)
   FILTER EVENT
   ... `Procesar el doble-clic
End if
```

Ejemplo

*Ver el ejemplo del comando **ON EVENT CALL**.*

Get assert enabled

Get assert enabled -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		True = las aserciones están activas, False = las aserciones están inactivas

Descripción

*El comando **Get assert enabled** devuelve True o False de acuerdo a si las aserciones están activas o no en el proceso actual. Para mayor información sobre aserciones, consulte la descripción del comando **ASSERT**.*

*Por defecto, las aserciones están activas pero pueden haber sido desactivadas utilizando el comando **SET ASSERT ENABLED**.*

GET LAST ERROR STACK

GET LAST ERROR STACK (arrayCodigos ; arrayCompInternos ; arrayTextos)

Parámetro	Tipo		Descripción
arrayCodigos	Array entero largo	⇦	Números de errores
arrayCompInternos	Array cadena	⇦	Códigos de componentes internos
arrayTextos	Array cadena	⇦	Texto de errores

Descripción

*El comando **GET LAST ERROR STACK** devuelve información sobre la pila de errores actual relacionada con el uso del kernel SQL de la aplicación 4D. Cuando una instrucción 4D provoca un error, la pila de errores actual contiene una descripción del error como también de todas las series de errores generadas. Por ejemplo, un error de tipo "disk full" provoca un error de escritura en el archivo luego un error en el comando de guardar registro: la pila por lo tanto contendrá tres errores. Si la última instrucción 4D no generó un error, la pila de errores actual está vacía.*

Este comando genérico puede utilizarse para procesar todo tipo de error que pueda ocurrir en la aplicación 4D.

***Nota:** sin embargo, para obtener la información detallada correspondiente a los errores generados por una fuente ODBC, será necesario utilizar el comando **SQL GET LAST ERROR**.*

*Este comando debe llamarse desde un método de llamada de errores instalado por el comando **ON ERR CALL**.*

La información se devuelve en tres arrays sincronizados:

- *arrayCodigos: este array recibe la lista de códigos de error generados.*
- *arrayCompInternos: este array contiene los códigos de los componentes internos asociados con cada error.*
- *arrayTextos: este array contiene el texto de cada error.*

*La lista de códigos de error y su texto se encuentra en la sección **Errores del motor SQL** del tema "Códigos de error".*

Method called on error

Method called on error -> Resultado

Parámetro	Tipo		Descripción
Resultado	Cadena		Nombre del método llamado por error

Descripción

*El comando **Method called on error** devuelve el nombre del método instalado por el comando **ON ERR CALL** para el proceso actual.*
Si no se ha instalado ningún método, se devuelve una cadena vacía ("").

Ejemplo

Este comando es particularmente útil en el contexto de componentes porque le permite cambiar temporalmente y luego restaurar los métodos de intercepción de errores:

```
$metActual:=Method called on error
ON ERR CALL("NuevoMetodo")
  ` Si el documento no puede abrirse, se genera un error
$ref:=Open document("MiDocumento")
  ` Reinstalación del método anterior
ON ERR CALL($metActual)
```

Method called on event

Method called on event -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Cadena*		*Nombre del método llamado por evento*

Descripción

*El comando **Method called on event** devuelve el nombre del método instalado por el comando **ON EVENT CALL**.*
Si no se ha instalado un ningún método, se devuelve una cadena vacía ("").

ON ERR CALL

ON ERR CALL (metodoError)

Parámetro	Tipo		Descripción
metodoError	Cadena	➡	Método de gestión de error a llamar o cadena vacía para desinstalar el método

Descripción

El comando **ON ERR CALL** instala el método de proyecto, cuyo nombre se pasa en metodoError, como método de intercepción de errores o método de gestión de errores.

Una vez que se instala un proyecto de control de errores, 4D llama al método cada vez que se produce un error durante la ejecución de un comando de lenguaje 4D.
El alcance de este comando es el proceso actual. Sólo puede tener un método de gestión de errores por proceso, pero puede tener diferentes métodos de gestión de errores para varios procesos. Para detener un método de gestión de errores, llame de nuevo **ON ERR CALL** y pase una cadena vacía en metodoError.

Notas:

- en el contexto de utilización de componentes, este comando se aplica sólo a la base actual. Si se genera un error de un componente, metodoError no se llama en la base local.
- Si **ON ERR CALL** se llama desde un proceso para el cual usted ha solicitado ejecución apropiativa (en modo compilado 64 bits), el compilador verifica si metodoError es hilo seguro y devuelve los errores si no es compatible con el modo apropiativo. Para más información, consulte la sección [#title id="8733"/].

Para desinstalar un método de gestión de errores, llame a **ON ERR CALL** de nuevo y pase la cadena vacía en metodoError.

Puede identificar errores leyendo la variable sistema Error, la cual contiene el número de código del error. Los códigos de errores se listan en el tema **Códigos de error**. Para mayor información, consulte la sección **Errores de sintaxis**. El valor de la variable Error es significativo sólo en el método de gestión de errores; si necesita el código del error en el método que provocó el error, copie la variable Error en su propia variable proceso. También puede acceder a las variables sistema Error method, Error line y Error formula las cuales contienen respectivamente, el nombre del método, el número de línea y el texto de la fórmula donde ocurrió el error (ver **Error, Error method, Error line**).

Puede utilizar el comando **GET LAST ERROR STACK** para obtener la secuencia de errores (la "pila" de errores) en el origen de la interrupción.

El método de gestión de errores debe tratar los errores de manera apropiada o mostrar un mensaje de error al usuario. Los errores pueden ser generados durante los procesos efectuados por:

- El motor de base de datos de 4D; por ejemplo, cuando guarda un registro provoca la violación de una regla de trigger.
- El entorno de 4D; por ejemplo, cuando no tienen suficiente memoria parar llenar un array.
- El sistema operativo en el cual se ejecuta la base; por ejemplo, disco lleno o errores de entrada/salida.

El comando **ABORT** puede utilizarse para terminar el proceso. Si no llama **ABORT** en el método instalado, 4D devuelve el método interrumpido y continúa la ejecución del método. Utilice el comando **ABORT** cuando la ejecución no puede recuperarse.

Si ocurre un error en el método de gestión de errores, 4D retoma el control de la gestión de errores. Por lo tanto, debe asegurarse de que el método de gestión de errores no pueda generar un error. Igualmente, no puede utilizar **ON ERR CALL** dentro del método de gestión de errores.

Ejemplo 1

El siguiente método de proyecto trata de crear un documento cuyo nombre se recibe como parámetro. Si no se puede crear el documento, el método de proyecto devuelve 0 (cero) o el código de error:

```
` Método de proyecto Crear doc
` Crear doc ( String ; Pointer ) -> Entero largo
` Crear doc ( DocName ; ->DocRef ) -> Código de error resultante

gError:=0
ON ERR CALL("IO MANEJADOR DE ERRORES")
$2->:=Create document($1)
ON ERR CALL("")
$0:=gError
```

El método de proyecto **IO_GESTION_ERRORES** es el siguiente:

```
` Método de proyecto IO_GESTION_ERRORES

gError:=Error ` Simplemente copie el código del error en la variable de proceso gError
```

Note la utilización de la variable proceso gError para obtener el código del error en el método de ejecución actual. Una vez estos métodos estén presentes en su base de datos, puede escribir:

```
  ` ...
C_TIME(vhDocRef)
$vlErrCode:=Crear doc($vsDocumentNombre;->vhDocRef)
If($vlErrCode=0)
  ` ...
   CLOSE DOCUMENT($vlErrCode)
Else
   ALERT("El documento no pudo ser creado, error de E/S "+String($vlErrCode))
End if
```

Ejemplo 2

*Ver el ejemplo de la sección **Arrays y memoria.***

Ejemplo 3

Mientras implementa un conjunto de operaciones complejas, puede terminar con varias subrutinas que necesiten diferentes métodos de gestión de errores. Sólo puede tener un método de gestión de errores por proceso, de manera que tiene dos opciones:
*- Mantener contacto con el actual cada vez que llama a **ON ERR CALL**, o*
*- Utiliza la variable array proceso (en este caso, asMetodoError) para "apilar" los métodos de gestión de errores y un método de proyecto (en este caso, **ON ERROR CALL**) para instalar y desinstalar los métodos de gestión de errores.*
Debe inicializar el array al comienzo de la ejecución del proceso:

```
  ` NO olvide inicializar el array al inicio
  ` del método de proceso (el método de proyecto que ejecuta el proceso)
ARRAY STRING(63;asMetodoError;0)
```

*Este es el método personalizado **ON ERROR CALL**:*

```
  ` Método de proyecto ON ERROR CALL
  ` ON ERROR CALL { ( Cadena) }
  ` ON ERROR CALL { ( Nombre del método ) }

C_STRING(63;$1;$MetodoError)
C_LONGINT($vlElem)

If(Count parameters>0)
   $MetodoError:=$1
Else
   $MetodoError:=""
End if

If($MetodoError#"")
   C_LONGINT(gError)
   gError:=0
   $vlElem:=1+Size of array(asMetodoError)
   INSERT IN ARRAY(asMetodoError;$vlElem)
   asMetodoError{$vlElem}:=$1
   ON ERR CALL($1)
Else
   ON ERR CALL("")
   $vlElem:=Size of array(asMetodoError)
   If($vlElem>0)
      DELETE FROM ARRAY(asMetodoError;$vlElem)
      If($vlElem>1)
         ON ERR CALL(asMetodoError{$vlElem-1})
      End if
   End if
End if
```

Luego, puede llamarlo de esta manera:

```
gError:=0
ON ERROR CALL("IO ERRORS") ` Instale el método de gestión de errores IO ERRORS
  ` ...
ON ERROR CALL("ALL ERRORS") ` Instale el método de gestión de errores ALL ERRORS
  ` ...
ON ERROR CALL ` Desinstale el método de gestión de errores ALL ERRORS y reinstale IO ERRORS
  ` ...
```

```
ON ERROR CALL ` Desinstale el método de gestión de errores IO ERRORS
` ...
```

Ejemplo 4

El siguiente método de gestión de errores ignora las interrupciones del usuario y muestra el texto del error:

```
 //Método de proyecto Show_errors_only
If(Error#1006) //esta no es una interrupción del usuario
   ALERT("The error "+String(Error)+" occurred. The code in question is: \""+Error formula+"\"")
End if
```

ON EVENT CALL

ON EVENT CALL (metodoEvento {; nombreProceso})

Parámetro	Tipo		Descripción
metodoEvento	Cadena	➡	Método de evento a llamar, o Cadena vacía para detener la interceptación de eventos
nombreProceso	Cadena	➡	Nombre del proceso

Descripción

El comando **ON EVENT CALL** instala el método, cuyo nombre se pasa en metodoEvento, como método de gestión de eventos.

Consejo: este comando necesita un nivel de conocimiento avanzado en programación. Generalmente, no necesita utilizar **ON EVENT CALL** para trabajar con eventos. Cuando utiliza formularios, 4D administra los eventos y los envía a los objetos y formularios apropiados.

Consejo: comandos tales como **GET MOUSE**, **Shift down,** etc., para obtener información sobre eventos. Estos comandos pueden llamarse desde los métodos de objeto para obtener la información que necesita sobre un evento involucrado con un objeto. Utilizarlos le ahorra la escritura de un algoritmo basado en una estructura de tipo **ON EVENT CALL**.

El alcance de este comando es la sesión de trabajo actual. Por defecto, el método se ejecuta en un proceso local separado. Sólo puede tener un método de gestión de eventos a la vez. Para detener un método de gestión de eventos, llame nuevamente **ON EVENT CALL** y pase una cadena vacía en metodoEvento.

Como el método de gestión de eventos se ejecuta en un proceso separado, está activo constantemente, incluso si ningún método de 4D se está ejecutando. Después de la instalación, 4D llama al método de gestión de eventos cada vez que ocurre un evento. Un evento puede ser un clic con el ratón o presionar una tecla.

El parámetro opcional nomProces da nombre al proceso creado por el comando **ON EVENT CALL**. Si nomProces comienza por el símbolo pesos ($), comienza un proceso local, lo cual generalmente es lo que usted quiere. Si omite el parámetro nomProces, 4D crea por defecto un proceso local llamado $Event Manager.

Advertencia: sea muy cuidadoso con lo que escribe en un método de gestión de eventos. NO llame comandos que generen eventos, de lo contrario será extremadamente difícil salir de la ejecución del método de gestión de eventos. La combinación de teclas **Ctrl+Mayús+Retroceso** (en Windows) o **Comando-Mayús-Control-Retroceso** (en Mac) le permite matar el proceso en el Gestor de eventos. Podría utilizar esta técnica para recuperar un método de gestión de eventos incontrolable (por ejemplo, uno que tenga eventos que disparen bugs).

En el método de gestión de eventos, puede leer las siguientes variables sistema **MouseDown**, **KeyCode**, **Modifiers**, **MouseX**, **MouseY** y **MouseProc**. Note que estas variables son variables proceso. Su alcance es por lo tanto el proceso de gestión de eventos. Cópielas en las variables interproceso si quiere que sus valores estén disponibles en otro proceso.

- La variable sistema **MouseDown** toma el valor 1 si el evento es un clic del ratón y 0 si no.
- La variable sistema **KeyCode** contiene el código del carácter digitado en el teclado o el código de una tecla de función. Consulte las secciones **Códigos Unicode** y **EXPORT TEXT** que listan los códigos de caracteres utilizados por 4D, así como también la sección **Códigos de teclas de función**.4D ofrece constantes predefinidas para los principales y teclas de funciones. En la ventana del explorador, busque los temas de estas constantes.
- La variable sistema Modifiers contiene el valor modificador. Indica si una tecla de modificación ha sido presionada cuando el evento ocurrió. Las siguientes teclas pueden ser detectadas:

Plataforma	Modificadores
Windows	Mayús, Bloqueo de mayúsculas, Alt, Ctrl,
Macintosh	Mayús, Bloqueo de mayúsculas, Alt u Opción, Comando, Control

Las teclas modificadoras no generan un evento por su cuenta; otra tecla o el botón del ratón también deben ser presionadas. La variable **Modifiers** es una variable Entero largo que contiene un campo de bits. 4D ofrece constantes predefinidas que especifican la posición del bit o la máscara de bits para cada tecla de modificación. Por ejemplo, para detectar si la tecla Mayús fue presionada para el evento, puede escribir:

```
If(Modifiers?? Shift key bit)  //Si la tecla Mayús fue presionada
```

o:

```
If((Modifiers & Shift key mask)#0)  //Si la tecla Mayús fue presionada
```

Puede utilizar una de las siguientes constantes, dependiendo de la tecla del modificador a probar en la plataforma, que se encuentra en el tema **Eventos (Modificadores)**:

Modificador	Constante
Mayúscula	Shift key bit / Shift key mask
Bloqueo de mayúsculas	Caps lock key bit / Caps lock key mask
Alt (también llamado Opción en OS X)	Option key bit / Option key mask
Ctrl en Windows	Command key bit / Command key mask
Ctrl en OS X	Control key bit / Control key mask
Comando en OS X	Command key bit / Command key mask
Clic derecho	Control key bit / Control key mask

- Las variables sistema MouseX y MouseY contienen las posiciones horizontal y vertical del clic del ratón, expresadas en el sistema de coordenadas locales de la ventana donde el clic se produjo. La esquina superior izquierda de la ventana es la

posición 0,0. Estas variables son significativas sólo cuando hay un clic del ratón.
- *La variable sistema **MouseProc** contiene el número de referencia del proceso en el cual ocurrió el evento (clic del ratón).*

***Importante:** las variables sistema **MouseDown**, **KeyCode**, **Modifiers**, **MouseX**, **MouseY** y **MouseProc** sólo contienen valores significativos en un método de gestión de eventos instalado con **ON EVENT CALL**.*

Ejemplo

Este ejemplo cancela la impresión si el usuario presiona las teclas Ctrl+punto. Primero, el método de gestión de eventos se instala. Luego aparece un mensaje, anunciando que el usuario puede cancelar la impresión. Si la variable interproceso vbWeStop es igual a True en el método de gestión de eventos, una caja de diálogo de alerta aparece para mostrar al usuario el número de registros que han sido impresos. Luego el método de gestión de eventos se desinstala:

```
PAGE SETUP
If(OK=1)
   ◇vbWeStop:=False
   ON EVENT CALL("GESTIÓN DE EVENTOS") ` Instala el método de gestión de eventos
   ALL RECORDS([Personas])
   MESSAGE("Para interrumpir la impresión presione Ctrl+Punto.")
   $vlNbRegistros:=Records in selection([People])
   For($vlRegistro;1;$vlNbRegistros)
      If(◇vbWeStop)
         ALERT("Printing cancelled at record "+String($vlRegistro)+" of "+String($vlNbRegistros))
         $vlRegistro:=$vlNbRegistros+1
      Else
         Print form([Personas];"Informe especial")
      End if
   End for
   PAGE BREAK
   ON EVENT CALL("") ` Desinstala el método de gestión de eventos
End if
```

Si se ha presionado la combinación Ctrl+punto, el método de gestión de eventos da a vbWeStop el valor True:

```
  ` Método de proyecto GESTIÓN DE EVENTOS
If((Modifiers?? Command key bit)&(KeyCode=Period))
   CONFIRM("¿Está seguro?")
   If(OK=1)
      ◇vbWeStop:=True
      FILTER EVENT ` NO olvide este llamado; de lo contrario 4D también obtendrá este evento
   End if
End if
```

*Note que este ejemplo utiliza **ON EVENT CALL** porque realiza un informe especial de impresión utilizando los comandos **PAGE SETUP**, **Print form** y **PAGE BREAK** en una estructura de tipo bucle **For...End for**.*
*Si imprime un informe utilizando **PRINT SELECTION**, NO necesita administrar los eventos que permiten al usuario interrumpir la impresión; **PRINT SELECTION** hace esto por usted.*

SET ASSERT ENABLED

SET ASSERT ENABLED (aser {; *})

Parámetro	Tipo		Descripción
aser	Booleano	➡	True = activar las aserciones, False = desactivar las aserciones
*	Operador	➡	Si se omite = el comando se aplica al conjunto de los procesos, Si se pasa = el comando se aplica al proceso actual únicamente

Descripción

El comando **SET ASSERT ENABLED** se utiliza para desactivar o reactivar las aserciones insertadas en el código 4D de la aplicación. Para mayor información sobre aseciones, consulte la descripción del comando **ASSERT**.

Por defecto, las aserciones añadidas en el programa están activas, en modo interpretado y en modo compilado. Este comando es útil para desactivarlas ya que su evaluación puede ser costosa en términos de tiempo de ejecución y usted también podría querer ocultarlas del usuario final de la aplicación. Por lo general, el comando **SET ASSERT ENABLED** puede utilizarse en el método base On Startup para activar o desactivar aserciones en función de si la aplicación está en modo "Prueba" o en modo "Producción".

Por defecto, el comando **SET ASSERT ENABLED** afecta todos los procesos de la aplicación. Para restringir el efecto del comando al proceso actual únicamente, pase el parámetro *.

Por favor tenga en cuenta que cuando las aserciones están desactivadas, las expresiones pasadas a los comandos **ASSERT** no se evalúan. Las líneas de código que llaman a **ASSERT** no tienen más efecto en el funcionamiento de la aplicación, ni en términos de comportamiento ni en términos de rendimiento.

Ejemplo

Desactivación de aserciones:

```
SET ASSERT ENABLED(False)
ASSERT(TestMethod) // TestMethod no se llamará ya que las aserciones están desactivadas
```

JSON

- *Presentación de comandos JSON*
- *JSON Parse*
- *JSON PARSE ARRAY*
- *JSON Resolve pointers*
- *JSON Stringify*
- *JSON Stringify array*
- *JSON TO SELECTION*
- *JSON Validate*
- *Selection to JSON*

Presentación de comandos JSON

Los comandos JSON permiten generar y analizar los objetos de lenguaje de formato JSON. Más particularmente, el formato JSON hace que sea posible el acceso a las bases 4D (datos y estructura) utilizando un navegador web.

*El soporte de objetos estructurados es una de las grandes novedades del lenguaje de 4D v14, destinado a facilitar el intercambio de datos estructurados. Gracias a los comandos del tema "JSON", 4D pueden trabajar directamente con objetos JSON. Sin embargo, 4D también puede trabajar con objetos "nativos" (cuya estructura se inspira en JSON), lo que permite el intercambio con todos los tipos de lenguaje. Para más información, consulte el capítulo **Objetos (Lenguaje)**.*

Introducción a JSON

"JSON o JavaScript Object Notation es un formato de datos basado en texto, genérico, derivado de la notación de los objetos del lenguaje ECMAScript." (fuente: Wikipedia). JSON es independiente de cualquier otro lenguaje, pero utiliza las convenciones que son familiares para los programadores que utilizan C++ o JavaScript, Perl, etc. Es un formato que es particularmente adecuado para el intercambio de datos.

Esta sección resume los principios de notación implementados en JSON. Para una descripción completa de este formato, consulte el siguiente sitio: www.json.org/index.html.

Sintaxis JSON

La sintaxis JSON está basada en los siguientes principios:

- *los datos consisten en pares de nombre/valor,*
- *los datos están separados por comas,*
- *los objetos están definidos por corchetes {},*
- *los arrays están definidos por corchetes [ ].*

Propiedades JSON

Los datos JSON se expresan en forma de pares nombre/valor (o llave/valor). Un par nombre/valor contiene un nombre de campo (entre comillas), luego dos puntos, seguido de un valor. Por ejemplo:

```
"Nombre":"Juan"
```

Para su información, este ejemplo equivale en JavaScript:

```
Nombre="Juan"
```

Tenga en cuenta que los nombres de propiedades son diacríticos y tienen en cuenta las mayúsculas y minúsculas. Si escribe "Nombre" en lugar de "nombre", obtiene un nuevo par nombre/valor.

Tipos de datos JSON

Los siguientes tipos de valores son soportados en JSON:

Tipo	Descripción	Comentarios
cadena	Todo carácter Unicode excepto " y \ Los valores, como los nombres de propiedades, están entre comillas ("), por ejemplo, "city":"París"	\ se utiliza para los caracteres de control: \" = comillas \\ = barra oblicua inversa \/ = barra \b = tecla de retroceso \f = formfeed \n = retorno de línea \r = retorno de carro \t = tabulación \u = cuatro cifras hexadécimales
número	Entero o número de punto flotante	Número similar a C o a Java, excepto que los formatos octal y hexadecimal no se utilizan
objeto	{ }	
array	[ ]	
booleano	true o false	
nulo	null	

Objetos JSON

Los objetos JSON se definen con corchetes y pueden contener un número indefinido de pares nombre/valor, por ejemplo:

```
{ "firstName":"John" , "lastName":"Doe" }
```

*Los objetos JSON se pueden almacenar y manipular en 4D por medio de variables **objeto** (**C_OBJECT**) y campos.*

Arrays JSON

Los arrays JSON se definen con corchetes. Cada array puede contener un número indefinido de objetos:

```
{ "employees": [ { "nombre":"Juan" , "apellido":"Pérez" }, { "nombre":"Ana" , "apellido":"Gómez" }, { "nombre":"Pedro" , "apellido":"Gónzalez" } ] }
```

Los arrays JSON se pueden almacenar y manipular en 4D por medio de variables de tipo ***Colección*** *(**C_COLLECTION**).*

Punteros JSON

*4D soporta y resuelve punteros JSON a través del comando **JSON Resolve pointers**. Un puntero JSON es una cadena que se puede utilizar para acceder a un campo o un valor de llave en particular en todo el documento JSON. Por convención, un URI que contiene un puntero JSON se puede encontrar en una propiedad objeto JSON que debe llamarse "$ ref".*

```
{ "$ref":<path>#<json_pointer> }
```

*Los punteros JSON se resuelven llamando el comando **JSON Resolve pointers** o automáticamente cuando se usa **Formularios dinámicos**.*

*Para más información, consulte la descripción del comando **JSON Resolve pointers**.*

Soporte Zona Horaria

Por defecto, cuando se convierten datos 4D desde JSON, se tiene en cuenta la zona horaria de la máquina en la que la conversión se lleva a cabo (de conformidad con JavaScript). Por ejemplo, en Francia (GMT+2), la conversión de !23/08/2013! da "2013-08-22T22:00:00Z" y viceversa.

*Puede cambiar este funcionamiento y no tener en cuenta la zona horaria, durante la implementación de los procedimientos de exportación, por ejemplo, utilizando el comando **SET DATABASE PARAMETER** (selector Dates inside objects). Note que este selector también puede influenciar la forma en que las fechas se almacenan en objetos.*

***Nota:** a partir de 4D v16 R6, las cadenas de fecha JSON en formato "AAAA-MM-DD" también pueden ser compatibles. Para más información, consulte la opción "Usar tipo de fecha en lugar de formato de fecha ISO en objetos" en **Página Compatibilidad**.*

*Para más información sobre cómo convertir fechas 4D/JSON, consulteo **Conversión de fechas JavaScript**.*

JSON Parse

JSON Parse (cadenaJSON {; tipo}{; *}) -> Resultado

Parámetro	Tipo		Descripción
cadenaJSON	Cadena	➡	Cadena en JSON a analizar
tipo	Entero largo	➡	Tipo en el cual convertir los valores
*	Operador	➡	Agrega la posición de la línea y el desplazamiento de cada propiedad si el valor devuelto es un objeto
Resultado	Mixed, Objeto	⬅	Valores extraídos de la cadena JSON

Descripción

*El comando **JSON Parse** analiza el contenido de una cadena con formato JSON y extrae los valores que puede almacenar en un campo o variable 4D. Este comando deserializa los datos JSON, realiza la acción inversa del comando **JSON Stringify**.*

En cadenaJSON, pase la cadena con formato JSON cuyo contenido desea analizar. Esta cadena debe tener el formato correcto, de lo contrario se genera un error de análisis.
***JSON Parse** por lo tanto puede ser utilizado para validar cadenas JSON.*

***Nota**: si utiliza punteros, debe llamar al comando **JSON Stringify** antes de llamar a **JSON Parse**.*

*Por defecto, si se omite el parámetro tipo, 4D intentará convertir el valor obtenido en el tipo de la variable o del campo que se utiliza para almacenar los resultados (si se ha definido). De lo contrario, 4D intenta deducir su tipo. También puede forzar la interpretación del tipo pasando el parámetro tipo: pase una de las siguientes constantes, disponibles en el tema **Tipos de campos y variables**:*

Constante	Tipo	Valor
Is Boolean	Entero largo	6
Is collection	Entero largo	42
Is date	Entero largo	4
Is longint	Entero largo	9
Is object	Entero largo	38
Is real	Entero largo	1
Is text	Entero largo	2
Is time	Entero largo	11

***Notas**:*

- *Los valores de tipo Real deben ser incluidos en el rango ± 10.421e±10*
- *En los valores de tipo de texto, todos los caracteres especiales deben ser escapados, incluyendo las comillas (ver ejemplos)*
- *Por defecto cuando se utiliza la constante Is date, el comando considera que una cadena fecha contiene una hora local y no GMT. Puede modificar esta configuración utilizando el selector Dates inside objects del comando **SET DATABASE PARAMETER**.*
- *A partir de 4D v16 R6, si la configuración de almacenamiento de fecha actual es "tipo fecha", las cadenas fecha JSON en formato "AAAA-MM-DD" son devueltas automáticamente como valores fecha por el comando **JSON Parse**. Para más información sobre esta configuración, consulte la opción "Utilizar tipo fecha en lugar de formato fecha ISO en objetos" en **Página Compatibilidad**.*
- *Un valor de tipo hora se pueden devolver a partir de números en cadenas. Por defecto, 4D considera que el valor es un número de segundos.*

*Si pasa el parámetro opcional * y si el parámetro cadenaJSON representa un objeto, el objeto devuelto contiene una propiedad adicional llamada __symbols que da la ruta, posición de línea y desplazamiento de línea de cada propiedad y sub-propiedad del objeto. Esta información puede ser útil para fines de depuración. La estructura de la propiedad __symbols es:*

__symbols:{//descripción del objeto myAtt.mySubAtt...:{ //ruta de la propiedad line:10, //número de línea de la propiedad offset:35 //offset de la propiedad desde el principio de la línea } }

***Nota:** el parámetro * se ignora si el valor devuelto no es del tipo objeto.*

Ejemplo 1

Ejemplos de conversiones simples:

```
C_REAL($r)
$r:=JSON Parse("42.17") //$r = 42,17 (Real)

C_LONGINT($el)
$el:=JSON Parse("120.13";Is longint) //$el=120

C_TEXT($t)
$t:=JSON Parse("\"Year 42\"";Is text) // $t="Year 42" (text)

C_OBJECT($o)
```

```
$o:=JSON Parse("{\"name\":\"john\"}")
 // $o = {"name":"john"} (4D object)

C_BOOLEAN($b)
$b:=JSON Parse("{\"manager\":true}";Is Boolean) // $b=true

C_TIME($h)
$h:=JSON Parse("5120";Is time) //$h=01:25:20
```

Ejemplo 2

Ejemplo de conversión de datos de tipo fecha:

```
$test:=JSON Parse("\"1990-12-25T12:00:00Z\"")
 // $test="1990-12-25T12:00:00Z"
C_DATE($date;$date2;$date3)
$date:=JSON Parse("\"2008-01-01T12:00:00Z\"";Is date)
 //$date=01/01/08
$date2:=JSON Parse("\"2017-07-13T23:00:00.000Z\"";Is date)
 //$date2=14/07/17 (Paris time zone)
SET DATABASE PARAMETER(Dates inside objects;String type without time zone)
$date3:=JSON Parse("\"2017-07-13T23:00:00.000Z\"";Is date)
 //$date3=13/07/17
```

Ejemplo 3

Si la configuración de almacenamiento de fecha actual es "tipo fecha", puede escribir:

```
C_OBJECT($o)
C_TEXT($json)
C_DATE($birthday)

$json:="{\"name\":\"Marcus\",\"birthday\":\"2017-10-16\"}"
$o:=JSON Parse($json)
$birthday:=$o.birthday
 //$birthday=16/10/17
```

Nota: *para más información sobre esta configuración, consulte la opción "Utilizar tipo fecha en lugar de formato de fecha ISO en objetos" en la* ***Página Compatibilidad****.*

Ejemplo 4

Este ejemplo muestra el uso combinado de los comandos ***JSON Stringify*** *y* ***JSON Parse****:*

```
C_TEXT($JSONContact)
C_OBJECT($Contact;$Contact2)
$Contact:=New object("name";"Monroe";"firstname";"Alan")

 // JSON Stringify: conversion of an object into a JSON string
$JSONContact:=JSON Stringify($Contact)

 // JSON Parse: conversion of JSON string into a new object
$Contact2:=JSON Parse($JSONContact)
```

Ejemplo 5

Usted desea crear una colección 4D desde un array JSON:

```
C_COLLECTION($myCol)
$myCol:=JSON Parse("[\"Monday\",10,\"Tuesday\",11,\"Wednesday\",12,false]")
```

Ejemplo 6

Usted desea analizar la siguiente cadena y obtener la posición de la línea y el desplazamiento de cada propiedad:

```
{ "alpha": 4552, "beta": [ { "echo": 45, "delta": "text1" }, { "echo": 52, "golf": "text2" } ] }
```

Puede escribir:

```
C_OBJECT($obInfo)
$obInfo=JSON Parse("json_string";Is object;*) //* para obtener la propiedad __symbols
 //en el objeto $obInfo devuelto
```

El objeto $obInfo contiene:

{alpha:4552, beta:[{echo:45,delta:text1},{echo:52,golf:text2}], __symbols:{alpha:{line:2,offset:4}, beta:{line:3,offset:4}, beta[0].echo:{line:5,offset:12}, beta[0].delta:{line:6,offset:12}, beta[1].echo:{line:9,offset:12}, beta[1].golf:{line:10,offset:12}}}

JSON PARSE ARRAY

JSON PARSE ARRAY (cadenaJSON ; array)

Parámetro	Tipo		Descripción
cadenaJSON	Cadena	➡	Cadena JSON a analizar
array	Array	⬅	Array que contiene el resultado del análisis de la cadena JSON

Descripción

*El comando **JSON PARSE ARRAY** analiza el contenido de una cadena con formato JSON y ubica los datos extraídos en el parámetro array. Este comando deserializa los datos JSON, realiza la acción inversa del comando **JSON Stringify array**.*

En cadenaJSON, pase la cadena con formato JSON cuyo contenido desea analizar. Esta cadena debe tener el formato correcto, de lo contrario se genera un error de análisis.

En array, pase un array que debe recibir los resultados del análisis.

***Nota:** a partir de 4D v16 R4, **JSON PARSE ARRAY** por lo general puede ser sustituida por una llamada a **JSON Parse** que devuelve una **colección**. Las colecciones se basan en arrays JSON y permiten almacenar datos de tipos mixtos, lo que ofrece más flexibilidad que los arrays.*

Ejemplo

En este ejemplo, los datos de los campos de los registros de una tabla se extraen y ubican en los arrays de objetos:

```
C_OBJECT($ref)
ARRAY OBJECT($sel;0)
ARRAY OBJECT($sel2;0)
C_TEXT(v_String)

OB SET($ref;"name";->[Company]Company Name)
OB SET($ref;"city";->[Company]City)

While(Not(End selection([Company])))
   $ref_company:=OB Copy($ref;True)
   APPEND TO ARRAY($sel;$ref_company)
 // $sel{1}={"name":"4D SAS","city":"Clichy"}
 // $sel{2}={"name":"MyComp","city":"Lyon"}
 // ...
   NEXT RECORD([Company])
End while

v_String:=JSON Stringify array($sel)
 // v_String= [{"name":"4D SAS","city":"Clichy"},{"name":"MyComp","city":"Lyon"}...]
JSON PARSE ARRAY(v_String;$sel2)
 // $sel2{1}={"name":"4D SAS","city":"Clichy"}
 // $sel2{2}={"name":"MyComp","city":"Lyon"}
 //...
```

JSON Resolve pointers

JSON Resolve pointers (objeto {; opciones}) -> Resultado

Parámetro	Tipo		Descripción
objeto	Objeto	➡	Objeto que contiene punteros JSON para resolver
		⬅	Objeto con punteros JSON resueltos (sólo si resultado es un objeto)
opciones	Objeto	➡	Opciones para la resolución de punteros
Resultado	Objeto	➡	Objeto que contiene el resultado del proceso

Descripción

*El comando **JSON Resolve pointers** resuelve todos los punteros JSON encontrados en el objeto, con respecto a la configuración de opciones (si existe).*

Los punteros JSON son particularmente útiles para:

- *anidar parte de un documento JSON externo o reutilizar una parte de un documento JSON en otros lugares en el mismo documento JSON, con el fin de factorizar la información,*
- *expresan una estructura cíclica en JSON,*
- *definir un objeto de plantilla que contiene las propiedades predeterminadas almacenadas en JSON.*

*Pase en el parámetro objeto un objeto que contiene punteros JSON que se deben resolver (para más información sobre la sintaxis del puntero JSON, consulte el párrafo **Definir punteros JSON** abajo).*

***Nota:** el objeto fuente se actualizará con el resultado de la resolución del puntero después de ejecutar el comando (excepto si el resultado no es un objeto, consulte abajo). Si desea conservar una versión original del objeto, puede considerar utilizar previamente **OB Copy**.*

Opcionalmente, puede pasar en opciones un objeto que contenga propiedades específicas que se utilizarán al resolver punteros. Se soportan las siguientes propiedades:

Propiedad	Tipo de valor	Descripción
rootFolder	Cadena	Ruta absoluta (utilizando la sintaxis 4D estándar) a la carpeta que se utilizará para resolver punteros relativos en el objeto. El valor predeterminado es la carpeta Recursos de la base.
merge	Booleano	Fusiona objetos con objetos puntero (true) en lugar de reemplazarlos (false). El valor predeterminado es false

Después de ejecutar el comando:

- *si el resultado de la resolución del puntero es un objeto, el objeto se actualiza y contiene el objeto resultante.*
- *si el resultado de la resolución del puntero es un valor escalar (es decir, un texto, un número...), el objeto se deja intacto y el valor resultante se devuelve en la propiedad "valor" del resultado de la función.*

En todos los casos, el comando devuelve un objeto que contiene las siguientes propiedades:

Propiedad	Tipo de valor	Descripción
value	Cualquiera	Resultado del procesamiento del comando en objeto. Si el resultado es un objeto, es igual al objeto de salida.
success	Booleano	true si todos los punteros han sido resueltos correctamen
errors	Colección	Colección de errores si los hay
errors[].code	Número	código del error
errors[].message	Cadena	mensaje de error
errors[].pointerURI	Cadena	valor del puntero
errors[].referredPath	Cadena	ruta completa del documento

Definir punteros JSON

JSON Pointer es un estándar que define una sintaxis de cadena que se puede utilizar para acceder a un campo o a un valor clave particular en todo el documento JSON. El estándar se ha descrito en RFC 6901.

Un puntero JSON es, estrictamente hablando, una cadena compuesta de partes separadas por '/'. Un puntero JSON normalmente se encuentra en un URI que especifica el documento en el que se resolverá el puntero. El carácter de fragmento "#" se utiliza en la URI para especificar el puntero JSON Por convención, se puede encontrar un URI que contenga un puntero JSON en una propiedad de objeto JSON que debe llamarse "$ ref".

{ "$ref":<path>#<json_pointer> }

***Nota**: 4D no soporta el carácter "-" como referencia a elementos de array inexistentes.*

Recursividad y resolución de ruta

Los punteros JSON se resuelven recursivamente, lo que significa que si un puntero resuelto también contiene punteros, se resuelven recursivamente y así sucesivamente, hasta que se resuelvan todos los punteros. En este contexto, todas las rutas de archivo encontradas en las URIs de puntero JSON pueden ser relativas o absolutas. Deben utilizar '/' como delimitador de ruta y se resuelven de la siguiente manera:

- *Una ruta relativa no debe comenzar con '/'. Se resuelve relativamente al documento JSON donde se ha encontrado la cadena de ruta de acceso,*

- *Una ruta absoluta comienza con '/'. Por razones de seguridad, sólo "/RESOURCES" se acepta como ruta absoluta y designa la carpeta de recursos de la base actual. Por ejemplo, "/RESOURCES/templates/myfile.json" apunta al archivo "myfile.json" que se encuentra en la carpeta de recursos de la base de datos actual.*

Notas:

- *La resolución del nombre distingue entre mayúsculas y minúsculas.*
- *4D no resuelve una ruta a un archivo json ubicado en la red (que empiece por "http/https").*

Ejemplo 1

Este ejemplo básico ilustra cómo un puntero JSON se puede definir y reemplazar en un objeto:

```
 // crear un objeto con algún valor
C_OBJECT($o)
$o:=New object("value";42)

 // crear el objeto puntero JSON
C_OBJECT($ref)
$ref:=New object("$ref";"#/value")

 // añadir el objeto puntero JSON como propiedad
$o.myJSONPointer:=$ref

 // resolverlo todo y verificar que el puntero se ha resuelto
C_OBJECT($result)
$options:=New object("rootFolder";Get 4D folder(Current resources folder);"merge";True)
$result:=JSON Resolve pointers($o;$options)
If($result.success)
   ALERT(JSON Stringify($result.value))
 //{"value":42,"myJSONPointer":42}
Else
   ALERT(JSON Stringify($result.errors))
End if
```

Ejemplo 2

Usted quiere reutilizar "billingAddress" como "shippingAddress" en el siguiente objeto JSON (llamado $oMyConfig):

```
{ "lastname": "Doe", "firstname": "John", "billingAddress": { "street": "95 S. Market Street", "city": "San Jose", "state": "California" }, "shippingAddress": { "$ref": "#/billingAddress" } }
```

Después de ejecutar este código:

```
$oResult:=JSON Resolve pointers($oMyConfig)
```

... se devuelve el siguiente objeto:

```
{ "success": true, "value": { "lastname": "Doe", "firstname": "John", "billingAddress": { "street": "95 S. Market Street", "city": "San Jose", "state": "California" }, "shippingAddress": { "street": "95 S. Market Street", "city": "San Jose", "state": "California" } } }
```

Ejemplo 3

Este ejemplo ilustra el efecto de la opción "fusionar". Usted desea editar los derechos de un usuario basándose en un archivo predeterminado.

```
{ "rights": { "$ref": "defaultSettings.json#/defaultRights", "delete": true, "id": 456 } }
```

El archivo defaultSettings.json contiene:

```
{ "defaultRights": { "edit": true, "add": false, "delete": false } }
```

Si ejecuta:

```
C_OBJECT($options)
$options:=New object("merge";False) //reemplazar contenidos
$oResult:=JSON Resolve pointers($oMyConfig;$options)
```

el valor resultante es exactamente el contenido del archivo defaultSettings.json:

```
{ "success": true, "value": { "rights": { "edit": true, "add": false, "delete": false } } }
```

Si ejecuta:

```
C_OBJECT($options)
$options:=New object("merge";True) //fusionar ambos contenidos
$oResult:=JSON Resolve pointers($oMyConfig;$options)
```

...el valor resultante es una versión modificada del objeto original:

{ "success": true, "value": { "rights": { "edit": true, "add": false, "delete": true, "id": 456 } } }

JSON Stringify

JSON Stringify (valor {; *}) -> Resultado

Parámetro	Tipo		Descripción
valor	Objeto, Mixed	➡	Datos a convertir en cadena JSON
*	Operador	➡	Mejorar el formato
Resultado	Texto	⬅	Cadena que contiene el texto JSON serializado

Descripción

*El comando **JSON Stringify** convierte el parámetro valor en una cadena JSON. Este comando realiza la acción opuesta del comando **JSON Parse**.*

Pase los datos a serializar en valor. Se pueden expresar en forma escalar (cadena, número, fecha u hora) o vía un objeto 4D o una colección.

***Nota**: las fechas 4D se convertirán en formato "aaaa-mm-dd" o "AAAA-MM-DDThh:mm:sssZ" según la configuración actual de la fecha de la base (ver la opción "Utilizar el tipo fecha en lugar del formato fecha ISO en los objetos" en **Página Compatibilidad**).*

*En el caso de un objeto o una colección, puede incluir todo tipo de valores (ver el párrafo **Tipos de datos JSON**), respetando las siguientes reglas de JSON:*

- *Los valores de tipo cadena deben ir entre comillas. Todos los caracteres Unicode pueden usarse excepto los caracteres especiales que deben ser precedidos por una barra oblicua invertida.*
- *Números: intervalo ±10.421e±10*
- *Booleano: cadenas "true " o " false"*
- *Punteros a un campo, variable o array (el puntero se evalúa al momento del stringify)*
- *Fecha: tipo texto en formato "aaaa-mm-dd" o "\"AAAA-MM-DDTHH:mm:ssZ"\", en función de los parámetros actuales de la base (ver arriba).*
- *Hora: tipo real (número de segundos por defecto)*

Notas:

- *Los atributos imagen se convierten a la siguiente cadena: "[objeto Imagen]".*
- *Los punteros a campos, variables o array son evaluados en el momento del stringify.*

*Puede pasar el parámetro opcional * con el fin de incluir caracteres con formato en la cadena resultante. Esta opción mejora la presentación de los datos JSON (pretty formatting).*

Ejemplo 1

Conversión de valores escalares:

```
$vc:=JSON Stringify("Eureka!") // "Eureka!"
$vel:=JSON Stringify(120) // "120"
$vh:=JSON Stringify(?20:00:00?) // "72000000" segundos desde la media noche
$vd:=JSON Stringify(!28/08/2013!) // "2013-08-27T22:00:00.000Z" (Paris timezone)
SET DATABASE PARAMETER(Dates inside objects;String type without time zone)
$vdd:=JSON Stringify(!28/08/2013!) // "2013-08-28T00:00:00.000Z"
```

Ejemplo 2

Conversión de una cadena que contiene caracteres especiales:

```
$s:=JSON Stringify("{\"name\":\"john\"}")
 // $s="{\\\"name\\\":\\\"john\\\"}"
$p:=JSON Parse($s)
 // $p={"name":"john"}
```

Ejemplo 3

*Ejemplos de serialización de un objeto 4D con y sin el parámetro *:*

```
C_TEXT($MyContact)
C_TEXT($MyPContact)
C_OBJECT($Contact;$Children)
OB SET($Contact;"lastname";"Monroe";"firstname";"Alan")
OB SET($Children;"firstname";"Jim";"age";"12")
OB SET($Contact;"children";$Children)
$MyContact:=JSON Stringify($Contact)
$MyPContact:=JSON Stringify($Contact;*)
```

```
   //$MyContact= {"lastname":"Monroe","firstname":"Alan","children":{"firstname":"John","age":"12"}}
   //$MyPContact= {\n\t"lastname": "Monroe",\n\t"firstname": "Alan",\n\t"children": {\n\t\t"firstname": "John",\n\t\t"age":
"12"\n\t}\n}
```

La ventaja de este formato es clara cuando el JSON se muestra en un área web:

- *Formato estándar:*

```
{"Name":"Monroe","firstname":"Alan","children":{"firstname":"John","age":12}}
```

- *Formato mejorado:*

```
{
        "Name": "Monroe",
        "firstname": "Alan",
        "children": {
                "firstname": "John",
                "age": 12
        }
}
```

Ejemplo 4

Ejemplo utilizando un puntero en una variable:

```
C_OBJECT($MyTestVar)
C_TEXT($name ;$jsonstring )
OB SET($MyTestVar;"name";->$name) // definición del objeto
 // $MyTestVar= {"name":"->$name"}

$jsonstring :=JSON Stringify($MyTestVar)
 // $jsonstring ={"name":""}"
 //...

$name:="Smith"
$jsonstring :=JSON Stringify($MyTestVar)
 //$jsonstring = "{"name" : "Smith"}"
```

Ejemplo 5

Serialización de un objeto 4D:

```
C_TEXT($varjsonTextserialized)
C_OBJECT($Contact)
OB SET($Contact;"firstname";"Alan")
OB SET($Contact;"lastname";"Monroe")
OB SET($Contact;"age";40)
OB SET($Contact;"phone";"[555-0100,555-0120]")

$varjsonTextserialized:=JSON Stringify($Contact)

 // $varjsonTextserialized = "{"lastname":"Monroe","phone":"[555-0100,
 // 555-0120]","age":40,"firstname":"Alan"}"
```

Ejemplo 6

Serialización de un objeto 4D que contiene un valor fecha (zona horaria de París). La cadena resultante depende de la configuración actual de la fecha de la base.

```
C_TEXT($varjsonTextserialized)
C_OBJECT($Contact)
OB SET($Contact;"name";"Smith";"birthday";!22/10/1975!)
$varjsonTextserialized:=JSON Stringify($Contact)
```

- *Si la opción "Utilizar tipo fecha en lugar de formato fecha ISO en objetos" no está seleccionada:*

```
"name":"Smith",
"birthday":"1975-10-21T22:00:00.000Z"
```

- *Si la opción "Utilizar tipo fecha en lugar de formato fecha ISO en objetos" está seleccionada:*

```
"name":"Smith",
"birthday":"1975-10-22"
```

***Nota**: para obtener más información sobre esta configuración, consulte **Página Compatibilidad**.*

Ejemplo 7

Conversión de una colección (zona horaria de París). La cadena resultante depende de la configuración actual de la fecha de la base de datos.

```
C_COLLECTION(myCol)
C_TEXT(myTxtCol)
myCol:=New collection(33;"mike";!28/08/2017!;False)
myTxtCol:=JSON Stringify(myCol)
```

- *Si la opción "Utilizar tipo fecha en lugar de formato fecha ISO en objetos" no está seleccionada:*

```
$myTxtCol="[33,"mike","2017-08-27T22:00:00.000Z",false]"
```

- *Si la opción "Utilizar tipo fecha en lugar de formato fecha ISO en objetos" está seleccionada:*

```
$myTxtCol="[33,"mike","2017-08-28",false]"
```

***Nota**: para más información sobre esta opción, consulte **Página Compatibilidad**.*

JSON Stringify array

JSON Stringify array (array {; *}) -> Resultado

Parámetro	Tipo		Descripción
array	Array texto, Array real, Array booleano, Array puntero, Array objeto	➡	Array cuyo contenido debe ser serializado
*	Operador	➡	Mejorar el formato
Resultado	Texto	↩	Cadena que contiene el array JSON serializado

Descripción

*El comando **JSON Stringify array** convierte el array array 4D en un array JSON serializado. Este comando realiza la acción inversa del comando **JSON PARSE ARRAY**.*

En array, pase un array 4D con los datos a serializar. Este array puede ser de tipo de texto, real, booleano, puntero u objeto.
***Nota:** si pasa una variable escalar o un campo en array, el comando devolverá simplemente el valor del parámetro entre "[ ]".*
*Puede pasar el parámetro opcional * para utilizar los caracteres de formato en la cadena resultante. Esta opción mejora la presentación de los datos JSON cuando se muestran en una página web (pretty formatting).*

Ejemplo 1

Conversión de un array texto:

```
C_TEXT($jsonString)
ARRAY TEXT($ArrayFirstname;2)
$ArrayFirstname{1}:="John"
$ArrayFirstname{2}:="Jim"
$jsonString :=JSON Stringify array($ArrayFirstname)

 // $jsonString = "["John","Jim"]"
```

Ejemplo 2

Conversión de un array texto que contiene números:

```
ARRAY TEXT($phoneNumbers;0)
APPEND TO ARRAY($phoneNumbers ;"555-0100")
APPEND TO ARRAY($phoneNumbers ;"555-0120")
$string :=JSON Stringify array($phoneNumbers)
 // $string = "["555-0100","555-0120"]"
```

Ejemplo 3

Conversión de un array objeto:

```
C_OBJECT($ref_john)
C_OBJECT($ref_jim)
ARRAY OBJECT($myArray;0)
OB SET($ref_john;"name";"John";"age";35)
OB SET($ref_jim;"name";"Jim";"age";40)
APPEND TO ARRAY($myArray ;$ref_john)
APPEND TO ARRAY($myArray ;$ref_jim)
$JsonString :=JSON Stringify array($myArray)
 // $JsonString = "[{"name":"John","age":35},{"name":"Jim","age":40}]"

 // Si desea visualizar el resultado en una página web,
 // pase el parámetro opcional *:
$JsonStringPretty :=JSON Stringify array($myArray;*)
```

```
[
  {
    "name": "John",
    "age": 35
  },
  {
    "name": "Jim",
    "age": 40
  }
]
```

Ejemplo 4

Conversión de una selección 4D en un array objeto:

```
C_OBJECT($jsonObject)
C_TEXT($jsonString)

QUERY([Company];[Company]Company Name="a@")
OB SET($jsonObject;"company name";->[Company]Company Name)
OB SET($jsonObject;"city";->[Company]City)
OB SET($jsonObject;"date";[Company]Date_input)
OB SET($jsonObject;"time";[Company]Time_input)
ARRAY OBJECT($arraySel;0)

While(Not(End selection([Company])))
   $ref_value:=OB Copy($jsonObject;True)
 // Si no los copia, los valores serán cadenas vacías
   APPEND TO ARRAY($arraySel;$ref_value)
 // Cada elemento contiene los valores seleccionados, por ejemplo:
 // $arraySel{1} = // {"company name":"APPLE","time":43200000,"city":
 // "Paris","date":"2012-08-02T00:00:00Z"}
   NEXT RECORD([Company])
End while

$jsonString:=JSON Stringify array($arraySel)
 // $jsonString = "[{"company name":"APPLE","time":43200000,"city":
 //"Paris","date":"2012-08-02T00:00:00Z"},{"company name":
 //"ALMANZA",...}]"
```

JSON TO SELECTION

JSON TO SELECTION (laTabla ; objetoJson)

Parámetro	Tipo		Descripción
laTabla	Tabla	➡	Puntero a la tabla 4D
objetoJson	Texto	➡	Cadena en JSON

Descripción

*El comando **JSON TO SELECTION** copia el contenido del array de objetos JSON jsonArray en la selección de registros de laTabla.*

El parámetro jsonArray es un texto representando un array de objetos JSON y contiene uno o más elementos. La sintaxis es del tipo:

"[{"attribute1":"value1","attribute2":"value2",...},...,{"attribute1":"valueN","attribute2":"valueN",...}]"

Si una selección existe para laTabla en el momento de la llamada, los elementos del array JSON se copian en los registros en función del orden del array y del orden de los registros. Si el número de elementos definidos en el array JSON es mayor que el número de registros de la selección actual, se crean nuevos registros. Los registros, ya sean nuevos o existentes, se guardan automáticamente.

***Nota:** este comando soporta los campos de tipo objeto: los datos JSON se convierten automáticamente.*

***Advertencia:** como **JSON TO SELECTION** reemplaza la información presente en los registros existentes, este comando se debe utilizar con prudencia.*

*Si un registro está bloqueado por otro proceso durante la ejecución del comando, no se modifica. Todos los registros bloqueados se ubican en el **Conjunto sistema LockedSet**. Después de la ejecución de **JSON TO SELECTION**, puede probar si el conjunto LockedSet contiene los registros que estaban bloqueados.*

Ejemplo

*Uso del comando **JSON TO SELECTION** para añadir los registros a la tabla [Company]:*

```
C_OBJECT($Object1;$Object2;$Object3;$Object4)
C_TEXT($ObjectString)
ARRAY OBJECT($arrayObject;0)

OB SET($Object1;"ID";"200";"Company Name";"4D SAS";"City";"Clichy")
APPEND TO ARRAY($arrayObject;$Object1)

OB SET($Object2;"ID";"201";"Company Name";"APPLE";"City";"Paris")
APPEND TO ARRAY($arrayObject;$Object2)

OB SET($Object3;"ID";"202";"Company Name";"IBM";"City";"London")
APPEND TO ARRAY($arrayObject;$Object3)

OB SET($Object4;"ID";"203";"Company Name";"MICROSOFT";"City";"New York")
APPEND TO ARRAY($arrayObject;$Object4)

$ObjectString:=JSON Stringify array($arrayObject)

 // $ObjectString = "[{"ID":"200","City":"Clichy","Company Name":"4D
 // SAS"},{"ID":"201","City":"Paris","Company Name":"APPLE"},{"ID":"202",
 //"City":"London","Company Name":"IBM"},{"ID":"203","City":"New
 //York","Company Name":"MICROSOFT"}]"

JSON TO SELECTION([Company];$ObjectString)
 // Usted crea 4 registros en la tabla [Company], llena los campos ID,
 //Nombres de empresa y ciudad
```

JSON Validate

JSON Validate (vJson ; vSchema) -> Resultado

Parámetro	Tipo		Descripción
vJson	Objeto	➡	Objeto JSON a validar
vSchema	Objeto	➡	Esquema JSON utilizado para validar objetos JSON
Resultado	Objeto	⮌	Estado de validación y errores (si los hay)

Descripción

*El comando **JSON Validate** verifica la conformidad del contenido JSON vJson con las reglas definidas en el esquema JSON vSchema. Si el JSON no es válido, el comando devuelve una descripción detallada de los errores.*

Pase en vJson un objeto JSON que contiene el contenido JSON a validar.

***Nota:** la validación de una cadena JSON consiste en comprobar que sigue las reglas definidas en un esquema JSON. Esto es diferente a comprobar que el JSON está bien formado, lo cual hace el comando **JSON Parse**.*

Pase en vSchema el esquema JSON a utilizar para la validación. Para mayor información sobre cómo crear un esquema JSON, puede consultar el sitio web json-schema.org.

***Nota:** para validar un objeto JSON, 4D utiliza la norma descrita en el documento JSON Schema Validation (este borrador aún se está escribiendo y puede evolucionar en el futuro). La implementación de 4D se basa en la versión 4 de este documento.*

*Si el esquema JSON no es válido, 4D devuelve un objeto **Null** y genera un error que puede detectarse por un método de llamada de error.*

***JSON Validate** devuelve un objeto que ofrece el estado de la validación. Este objeto puede contener las siguientes propiedades:*

Nombre de propiedad	Tipo	Descripción
success	Booleano	True si vJson está validado, false en caso contrario. Si es false, la propiedad errors también se devuelve
errors	Colección de objetos	Lista de objetos de error en caso de que vJson no esté validado (ver abajo)

Cada objeto de error de la colección errors contiene las siguientes propiedades:

Nombre de propiedad	Tipo	Descripción
code	Número	Código de error
jsonPath	Cadena	Ruta JSON que no se puede validar en vJson
line	Número	Número de línea del error en el archivo JSON. Esta propiedad se llena si el JSON ha sido analizado por **JSON Parse** con el parámetro *. De lo contrario, la propiedad se omite.
message	Cadena	Mensaje de error
offset	Número	Desplazamiento de línea del error en el archivo JSON. Esta propiedad se llena si el JSON ha sido analizado por **JSON Parse** con el parámetro *. De lo contrario, la propiedad se omite.
schemaPaths	Cadena	Ruta JSON en el esquema que causa el error de validación

Gestión de errores

Se pueden devolver los siguientes errores:

Código	Palabra clave JSON	Mensaje[#table]
2	multipleOf	Error al validar contra la llave'multipleOf'.
3	maximum	El valor dado no debe ser mayor que el especificado en el esquema ("{s1}").
4	exclusiveMaximum	El valor dado debe ser menor que el especificado en el esquema ("{s1}").
5	minimum	El valor dado no debe ser menor que el especificado en el esquema ("{s1}").
6	exclusiveMinimum	El valor dado debe ser mayor que el especificado en el esquema ("{s1}").
7	maxLength	La cadena es más larga que la especificada en el esquema.
8	minLength	La cadena es más corta que la especificada en el esquema.
9	pattern	La cadena "{s1}" no coincide con el patrón del esquema: {s2}.
10	additionalItems	Error al validar un array. JSON contiene más elementos que los especificados en el esquema.
11	maxItems	El array contiene más elementos que los especificados en el esquema.
12	minItems	El array contiene menos elementos que los especificados en el esquema.
13	uniqueItems	Error al validar un array. Los elementos no son únicos. Otra instancia de "{s1}" ya está en el array.
14	maxProperties	El número de propiedades es mayor que el especificado en el esquema.
15	minProperties	El número de propiedades es menor que el especificado en el esquema.
16	required	Falta la propiedad requerida "{s1}".
17	additionalProperties	No hay propiedades adicionales permitidas por el esquema. La(s) propiedad(es) {s1} debe(n) ser eliminada(s).
18	dependencies	La propiedad "{s1}" requiere la propiedad "{s2}".
19	enum	Error al validar contra la llave 'enum'. "{s1}" no coincide con ningún elemento enum del esquema.
20	type	Tipo incorrecto. El tipo esperado es: {s1}
21	oneOf	El JSON coincide con más de un valor.
22	oneOf	El JSON no coincide con ningún valor.
23	not	El JSON es válido contra el valor de 'not'.
24	format.	La cadena no coincide ("{s1}")

Ejemplo

Usted desea validar un objeto JSON con un esquema y obtener la lista de errores de validación, si los hay, y guardar líneas de error y mensajes en una variable texto:

```
C_OBJECT($oResult)
$oResult:=JSON Validate(JSON Parse(myJson;*);mySchema)
If($oResult.success) //validación exitosa
   ...
Else //Validación fallida
   C_LONGINT($vLNbErr)
   C_TEXT($vTerrLine)
   $vLNbErr:=$oResult.errors.length ///obtener el número de error(es)
   ALERT(String($vLNbErr)+" validation error(s) found.")
   For($i;0;$vLNbErr)
      $vTerrLine:=$vTerrLine+$oResult.errors[$i].message+" "+String($oResult.errors[$i].line)+Carriage return
   End for
End if
```

Nota: *este ejemplo requiere que la notación de objeto esté activada (ver el párrafo* ***Página Compatibilidad****).*

Selection to JSON

Selection to JSON (laTabla {; elCampo}{; elCampo2 ; ... ; elCampoN}{; template}) -> Resultado

Parámetro	Tipo		Descripción
laTabla	Tabla	➡	Tabla a serializar
elCampo	Campo	➡	Campo(s) cuyo(s) contenidos deben ser serializados
template	Objeto	➡	Objeto para la selección de etiquetas y de campos
Resultado	Texto	⮌	Cadena que contiene el array JSON serializado

Descripción

*El comando **Selection to JSON** devuelve una cadena que contiene un array JSON con tantos elementos como registros hay en la selección actual de laTabla. Cada elemento del array es un objeto JSON que contiene las etiquetas y los valores de los campos de la selección.*

Si sólo pasa el parámetro laTabla, el comando incluye en el array JSON, los valores de todos los campos de la tabla que se pueden expresar en JSON. Los campos tipo BLOB e imagen se ignoran.

Si no desea incluir todos los campos de laTabla, puede utilizar el parámetro elCampo o el parámetro plantilla:

- *unCampo: pase uno o más campos en este parámetro. Sólo los valores de los campos definidos se incluyen en el array JSON.*
- *plantilla: pase un objeto 4D que contenga uno o más pares nombre/valor donde el nombre puede ser todo nombre de atributo valido y el valor contiene un puntero a un campo a incluir. Esta sintaxis le permite personalizar las etiquetas de campos en el array JSON.*

Este comando soporta campos de tipo Objeto: los datos de estos campos se convierte automáticamente en formato JSON (los valores de atributo imagense convierten como cadenas "(objeto Imagen)". Tenga en cuenta que la siguiente instrucción 4D será interpretado como "producir JSON a partir de todos los valores de campoObjeto en la selección actual de la tabla":

```
Selection to JSON([aTable];objectField)
```

***Nota:** después de un llamado a **Selection to JSON**, la selección actual no se modifica, pero el registro actual no se carga y podría haber cambiado (el último registro de la selección actual es entonces el registro actual). Después del comando **Selection to JSON**, utilice los comandos **LOAD RECORD** en combinación con **GOTO SELECTED RECORD** (si es necesario) utilice los valores de los campos en el registro actual.*

Ejemplo 1

Quiere crear una cadena JSON que represente esta selección:

HierList - Members: 6 of 6

Last name :	First name :	Address :	City :	Zip Code :
Durant	Mark	25 Park St	Pittsburgh	15205
Smith	John	24 Philadelphia Ave	Dallas	75203
Anderson	Adeline	37 Market St	Cincinnati	45205
Peterson	Paul	32 South Main St	Dallas	75203
Harper	Harry	233 Southport Ave	Cincinnati	45206
Trace	Sandra	332 Court St	Pittsburgh	15205

1) Desea incluir los valores de todos los campos de la tabla [Members]:

```
$jsonString :=Selection to JSON([Members])
 // $jsonString =[{"LastName":"Durant","FirstName":"Mark","Address":
 //"25 Park St","Zip code":"15205","City":"Pittsburgh"},{"LastName":
 //"Smith","FirstName":"John","Address":"24 Philadelphia Ave","Zip code":
 //"75203","City":"Dallas"},{"LastName":"Anderson","FirstName"
 //:"Adeline","Address":"37 Market St","Zip code":"45205","City":"Cincinnati"},...]
```

2) Desea reducir la selección y sólo incluir dos campos en la cadena JSON utilizando la sintaxis basada en los campos:

```
QUERY([Members];[Members]LastName="A@")
$jsonString :=Selection to JSON([Members];[Members]LastName;[Members]City)
 // $jsonString = [{"LastName":"Anderson","City":"Cincinnati"},{"LastName":"Albert","City":"Houston"}]
```

3) Sólo desea incluir un campo en la cadena JSON y utilizar una etiqueta diferente.
Puede utilizar la sintaxis template:

```
C_OBJECT($template)
OB SET($template;"Member";->[Members]LastName) //etiqueta personalizada y un campo sencillo
ALL RECORDS([Members])
$jsonString :=Selection to JSON([Members];$template)
 // $jsonString = [{"Member":"Durant"},{"Member":"Smith"},{"Member":"Anderson"},
 // {"Member":"Albert"},{"Member":"Leonard"},{"Member":"Pradel"}]
```

Ejemplo 2

Puede utilizar la sintaxis template para exportar campos de tablas diferentes:

```
C_OBJECT($template)
C_TEXT($jsonString)
OB SET($template;"Last name";->[Emp]LastName)
OB SET($template;"First name";->[Emp]FirstName)
OB SET($template;"Company";->[Company]LastName) //etiqueta personalizada de lo contrario conflicto con el campo [Emp]LastName
ALL RECORDS([Emp])
SET FIELD RELATION([Emp]UUID_Company;Automatic;Do not modify)
$jsonString:=Selection to JSON([Emp];$template)
SET FIELD RELATION([Emp]UUID_Company;Structure configuration;Do not modify)
```

LDAP

- *Presentación de los comandos LDAP*
- *LDAP LOGIN*
- *LDAP LOGOUT*
- *LDAP Search*
- *LDAP SEARCH ALL*

Presentación de los comandos LDAP

Los comandos del tema "LDAP" permiten a su aplicación 4D conectarse a un directorio de empresa tal como MS Active Directory con la ayuda de LDAP. Puede acceder a los datos del servidor y efectuar búsquedas.

***Nota:** LDAP o Lightweight Directory Access Protocol es un estándar para el acceso y el mantenimiento de los servicios de información distribuidos. Para más información, consulte la página Wikipedia en LDAP o la página principal OpenLDAP Software.*

Los comandos LDAP le permiten:

- *utilizar el inicio de sesión y la contraseña de Windows para permitir el acceso a su aplicación 4D (si utiliza el directorio MS Active) de manera que el usuario final sólo tiene que recordar una única contraseña,*
- *buscar el directorio de la empresa para recuperar la información del usuario, como nombre completo, dirección de correo electrónico, número de teléfono, edificio, grupos a los que pertenece, etc.*

*En 4D, una conexión LDAP se abre utilizando **LDAP LOGIN**. A continuación, se une al proceso 4D y debe ser cerrado utilizando **LDAP LOGOUT**, o cuando el proceso termina su ejecución.*

Glosario

Aquí está una lista de los principales acrónimos utilizados en el entorno LDAP:

Acrónimo	***Definición***
LDAP	*Lightweight Directory Access Protocol*
AD	*Active Directory. AD es una base de servicios de directorio implementada por Microsoft, y LDAP es uno de los protocolos de intercambio.*
CN	*Common Name, por ejemplo "John Doe"*
DN	*Distinguished Name, por ejemplo "cn=John Doe,ou=users,dc=example,dc=com"*
SAM-Account-Name	*nombre de inicio de sesión para la conexión AD, por ejemplo "jdoe"*
OU	*Organizational unit, grupos del árbol servidor*
DC	*Domain components, raíz y primeras ramas del árbol servidor*
uid	*User identifier*

LDAP LOGIN

LDAP LOGIN (url ; login ; password {; digest})

Parámetro	Tipo		Descripción
url	Cadena	➡	URL del servidor LDAP al cual conectarse
login	Cadena	➡	Cuenta del usuario
password	Cadena	➡	Contraseña del usuario
digest	Entero largo	➡	0 = enviar contraseña en digest MD5 (por defecto), 1 = enviar contraseña sin encripción

Descripción

*El comando **LDAP LOGIN** abre una conexión de sólo lectura en el servidor LDAP especificado por el parámetro url con los identificadores login y contraseña suministrados. Si es aceptado por el servidor, esta conexión se utiliza para todas las búsquedas LDAP efectuadas posteriormente en el proceso actual hasta que el comando **RuntimeVLWinFolder** se ejecute (o hasta que el proceso se cierre).*

En url, pase el URL completo del servidor LDAP al cual conectarse, incluyendo el esquema y el puerto (389 por defecto). Este parámetro debe ser compatible con la rfc2255.

Puede abrir conexiones seguras a través de TLS utilizando una mediante el uso de una url que empieza comience por "ldaps" y que utilice un número de puerto específico (por ejemplo "ldaps://svr.ldap.acme.com:1389"). El servidor LDAP debe tener un certificado SSL (al menos para Microsoft Active Directory). Es muy recomendable utilizar una conexión TLS cuando se envía la contraseña en texto plano (ver más abajo).

***Nota:** si pasa una cadena vacía en el parámetro url, el comando intentará conectarse al servidor LDAP predeterminado disponible en el dominio; (esta funcionalidad está diseñada para propósitos de prueba solamente, por razones de rendimiento no debería ser utilizada en producción).*

En login, pase la cuenta de usuario en el servidor LDAP, y en contraseña, pase la contraseña de usuario. Por defecto, el login puede ser una de las siguientes cadenas de inicio de sesión, dependiendo de la configuración del servidor LDAP:

- *un Distinguished Name (DN), por ejemplo "CN=John Smith,OU=users,DC=example,DC=com"*
- *un nombre de usuario (CN), por ejemplo "CN=John Smith"*
- *una dirección de correo electrónico, por ejemplo "johnsmith@4d.fr"*
- *un SAM-Account-Name, por ejemplo "jsmith".*

Tenga en cuenta que los valores aceptados para el login están relacionadas con el modo de transmisión de la contraseña definido por el parámetro digest. Por ejemplo, en una configuración por defecto de MS Active Directory:

- *Cuando el modo de transmisión es LDAP password MD5, el único valor aceptado para un inicio de sesión es el SAM-Account-Name.*
- *Cuando el modo de transmisión es LDAP password plain text (texto plano), el parámetro login puede ser DN, CN o una dirección de correo electrónico. Un SAM-Account-Name también se acepta, pero sólo cuando es precedido por el nombre de dominio (por ejemplo, (for example "dc-acme.com/jsmith").*

*El parámetro digest le permite modificar la forma en que la contraseña se transmite por la red. Puede utilizar una de las siguientes constantes, ubicadas en el tema **LDAP**:*

Constante	Tipo	Valor	Comentario
LDAP password MD5	Entero largo	0	(Por defecto) Enviar contraseña encriptada en MD5
LDAP password plain text	Entero largo	1	Envío de contraseña sin encripción (conexión TLS recomendada)

Por defecto, la contraseña se transmite en digest MD5. Pase LDAP password plain text si es necesario, por ejemplo, si desea utilizar diferentes valores de tipo de inicio de sesión con el servidor LDAP. En un entorno de producción, se recomienda utilizar una conexión TLS para la url.

***Nota:** la autenticación con una contraseña vacía permite introducir el modo de conexión anónima (si está autorizado por el servidor LDAP). Sin embargo, en este modo, se pueden generar errores si se intenta realizar cualquier operación que no esté permitida en este modo específico.*

Si los parámetros de inicio de sesión son válidos, una conexión con el servidor LDAP se abre en el proceso 4D. Luego puede buscar y recuperar información utilizando los comandos LDAP.

*No olvide llamar al comando **RuntimeVLWinFolder** cuando la conexión al servidor LDAP ya no sea necesaria.*

Ejemplo 1

Usted quiere conectarse a un servidor LDAP y efectuar una búsqueda:

```
ARRAY TEXT($_tabAttributes;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes;"phoneNumber")
LDAP LOGIN("ldap://srv.dc.acme.com:389";"John Smith";"qrnSurBret2elburg")
$vfound:=LDAP Search("OU=UO_Users,DC=ACME,DC=com";cn=John Doe";LDAP all levels;$_tabAttributes)
LDAP LOGOUT  //no olvide desconectarse
```

Ejemplo 2

Este ejemplo intenta conectarse a una aplicación:

```
ON ERR CALL("ErrHdlr") //gestión de errores
errOccured:=False
errMsg:=""
If(ppBindMode=1) //si contraseña es modo por defecto
    LDAP LOGIN(vUrlLdap;vUserCN;vPwd;LDAP password MD5)
Else
    LDAP LOGIN(vUrlLdap;vUserCN;vPwd;LDAP password plain text)
End if

Case of
    :(Not(errOccured))
        ALERT("Ahora está conectado a su servidor LDAP.   ")

    :(errOccured)
        ALERT("Errores en sus parámetros")
End case

LDAP LOGOUT
ON ERR CALL("")
```

LDAP LOGOUT

LDAP LOGOUT
Este comando no requiere parámetros

Descripción

*El comando **LDAP LOGOUT** cierra la conexión con un servidor LDAP en el proceso actual (si aplica). Si no hay conexión, se devuelve el error 1003 indicando que no está conectado.*

LDAP Search

LDAP Search (dnRootEntry ; filtro {; alcance {; atributos {; atributosEnArray}}}) -> Resultado

Parámetro	Tipo		Descripción
dnRootEntry	Cadena	➡	Distinguished Name del elemento raíz donde la búsqueda se inicia
filtro	Cadena	➡	Filtro de búsqueda LDAP
alcance	Cadena	➡	Campo de acción de la búsqueda: "base" (por defecto), "one", o "sub"
atributos	Array texto	➡	Atributo(s) a recuperar
atributosEnArray	Array booleano	➡	True = forzar el retorno de los atributos como array; False = forzar el retorno de los atributos como una variable simple
Resultado	Objeto	↩	Atributos llave/valor

Descripción

*El comando **LDAP Search** busca la primera ocurrencia que coincida con los criterios definidos en el servidor LDAP objetivo. Este comando debe ser ejecutado dentro de una conexión a un servidor LDAP abierta con **RuntimeVLIncludeIt**; de lo contrario se devuelve un error 1003.*

En dnRootEntry, pase el Distinguished Name del elemento raíz del servidor LDAP; la búsqueda se iniciará a partir de este elemento.

En filtro, pase el filtro de búsqueda LDAP a aplicar. La cadena filtro debe ser compatible con rfc2225. Puede pasar una cadena vacía "" para no filtrar la búsqueda; el "" se soporta para buscar subcadenas.*

En alcance, pase una de las siguientes constantes del tema "LDAP":

Constante	Tipo	Valor	Comentario
LDAP all levels	Cadena	sub	Buscar en el elemento raíz definido por dnRootEntry y en todas las ramas siguientes
LDAP root and next	Cadena	one	Buscar en el nivel de entrada raíz definido por dnRootEntry y en las entradas directamente posteriores en un nivel
LDAP root only	Cadena	base	Buscar únicamente en el elemento raíz definido por dnRootEntry (se omite por defecto)

En atributos, pase un array texto que contiene la lista de todos los atributos LDAP a recuperar a partir de las entradas encontradas. Por defecto, si se omite este parámetro, todos los atributos se recuperan.

***Nota:** tenga en cuenta que los nombres de atributos LDAP distinguen entre mayúsculas y minúsculas. Para más información sobre los atributos LDAP, puede consultar esta página que lista todos los atributos disponibles para MS Active directory.*

Por defecto, el comando devuelve los atributos en forma de colección si se encuentran varios resultados o en forma de variable si se encuentra un solo resultado. El parámetro opcional atributosEnArray permite "forzar" el formato de los atributos devueltos en colección o en variable para cada atributo definido:

- *Cuando pase **true** en un elemento, el elemento correspondiente del parámetro atributos será devuelto en una colección. Si se encuentra un solo valor, el comando devuelve una colección con un solo elemento.*
- *Cuando pase **false** en un elemento, el elemento correspondiente del parámetro atributos será devuelto en una variable simple. Si se encuentran varias entradas, el comando devuelve sólo el primer elemento.*

Ejemplo 1

Usted desea conseguir el número de teléfono del usuario "smith" en el directorio de la empresa:

```
ARRAY TEXT($_tabAttributes;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes;"phoneNumber")
LDAP LOGIN($url;$dn;$pwd)
$filter:="cn=*smith*"
$vfound:=LDAP Search($dnSearchRootEntry;$filter;LDAP all levels;$_tabAttributes)
LDAP LOGOUT
```

Ejemplo 2

Queremos obtener una array de todas las entradas que se encuentran en el atributo "memberOf":

```
C_OBJECT($entry)
ARRAY TEXT($_tabAttributes;0)
ARRAY BOOLEAN($_tabAttributes_asArray;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes_asArray;False)
APPEND TO ARRAY($_tabAttributes;"memberOf")
APPEND TO ARRAY($_tabAttributes_asArray;True)

LDAP LOGIN($url;$login;$pwd;LDAP password plain text)
$entry:=LDAP Search($dnSearchRootEntry;"cn=adrien*";LDAP all levels;$_tabAttributes;$_tabAttributes_asArray)
```

```
LDAP LOGOUT

ARRAY TEXT($_arrMemberOf;0)
OB GET ARRAY($entry;"memberOf";$_arrMemberOf)
 // en $_arrMemberOf tenemos un array que contiene todos los grupos de la entrada
```

LDAP SEARCH ALL

LDAP SEARCH ALL (dnRootEntry ; arrResult ; filtro {; alcance {; atributos {; atributosenArray}}})

Parámetro	Tipo		Descripción
dnRootEntry	Cadena	➡	Distinguished Name del elemento raíz donde iniciar la búsqueda
arrResult	Array objeto	⬅	Resultado de la búsqueda
filtro	Cadena	➡	Filtro de búsqueda LDAP
alcance	Cadena	➡	Alcance de la búsqueda: "base" (por defecto), "one", o "sub"
atributos	Array texto	➡	Atributos a recuperar
atributosenArray	Array booleano	➡	True = forzar el retorno de los atributos como array; false = forzar el retorno de los atributos como variables simples

Descripción

*El comando **LDAP SEARCH ALL** busca todas las ocurrencias que coinciden con los criterios definidos en el servidor LDAP objetivo. Este comando debe ser ejecutado dentro de una conexión a un servidor LDAP abierta con **LDAP LOGIN**; de lo contrario se devuelve un error 1003.*

Tenga en cuenta que los servidores LDAP generalmente imponen un número máximo de entradas que se pueden recibir de una búsqueda. Por ejemplo, el directorio de Microsoft Active limita este número a 1.000 entradas por defecto.

En dnRootEntry, pase el Distinguished Name del elemento raíz del servidor LDAP; la búsqueda se iniciará a partir de este elemento.

En tabResult, pase un array objeto que se llenará con todas las entradas coincidentes; en este array, cada elemento es un objeto que contiene los pares atributo/valor devueltos por una entrada coincidente. Puede utilizar el parámetro atributos para definir los parámetros a devolver.

En filtro, pase el filtro de búsqueda LDAP a aplicar. La cadena filtro debe ser compatible con rfc2225. Puede pasar una cadena vacía "" para no filtrar la búsqueda; el "" se soporta para buscar subcadenas.*

En alcance, pase una de las siguientes constantes del tema "LDAP":

Constante	Tipo	Valor	Comentario
LDAP root only	Texto	"base"	Buscar sólo en el elemento raíz definido por dnRootEntry (por defecto si se omite)
LDAP root and next	Texto	"one"	Buscar en el elemento la raíz definido por dnRootEntry y en las entradas posteriores directamente en un nivel
LDAP all levels	Texto	"sub"	Buscar en el elemento raíz definido por dnRootEntry y en todas las entradas posteriores

En atributos, pase un array texto que contiene la lista de todos los atributos LDAP a recuperar a partir de las entradas encontradas. Por defecto, si se omite este parámetro, todos los atributos se recuperan.

***Nota:** tenga en cuenta que los nombres de atributos LDAP distinguen entre mayúsculas y minúsculas. Para más información sobre los atributos LDAP, puede consultar esta página que lista todos los atributos disponibles para MS Active directory.*

Por defecto, el comando devuelve atributos como un array si se encuentran varios resultados, o como una variable si se encuentra un solo resultado. Los parámetros opcionales atributosEnArray permiten "forzar" el formato de los atributos devueltos en un array o como una variable para cada atributo definido:

- *Cuando pase **true** en un elemento, el elemento correspondiente del parámetro atributos será devuelto en un array. Si se encuentra un solo valor, el comando devuelve un array con un solo elemento.*
- *Cuando pase **false** en un elemento, el elemento correspondiente del parámetro atributos será devuelto en una variable simple. Si se encuentran varias entradas, el comando devuelve sólo el primer elemento.*

Ejemplo 1

Queremos obtener el número de teléfono de todos los usuarios con nombre "smith" en el directorio de la empresa:

```
ARRAY TEXT($_tabAttributes;0)
ARRAY BOOLEAN($_tabAttributes_asArray;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes_asArray;False)
APPEND TO ARRAY($_tabAttributes;"telephoneNumber")
APPEND TO ARRAY($_tabAttributes_asArray;False)
ARRAY OBJECT($_entry;0)

LDAP LOGIN($url;$myLogin;$pwd)
$filter:="cn=*smith*"
LDAP SEARCH ALL($dnSearchRootEntry;$_entry;$filter;LDAP all levels;$_tabAttributes)
LDAP LOGOUT


 //$_entry contendrá por ejemplo
 // $_entry{1} = {"cn":"John Smith","telephoneNumber":"01 40 87 00 00"}
 // $_entry{2} = {"cn":"Adele Smith","telephoneNumber":"01 40 87 00 01"}
 // $_entry{3} = {"cn":"Adrian Smith","telephoneNumber":"01 23 45 67 89"}
 // ...
```

Ejemplo 2

Estos ejemplos ilustran el uso del parámetro atributosEnArray:

```
ARRAY OBJECT($_entry;0)
ARRAY TEXT($_tabAttributes;0)
ARRAY BOOLEAN($_tabAttributes_asArray;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes_asArray;False)
APPEND TO ARRAY($_tabAttributes;"memberOf")
APPEND TO ARRAY($_tabAttributes_asArray;True)

LDAP LOGIN($url;$login;$pwd;LDAP password plain text)
LDAP SEARCH ALL($dnSearchRootEntry;$_entry;$filter;LDAP all levels;$_tabAttributes;$_tabAttributes_asArray)
LDAP LOGOUT

ARRAY TEXT($_arrMemberOf;0)
OB GET ARRAY($_entry{1};"memberOf";$_arrMemberOf)
 // en $_arrMemberOf tenemos un array que contiene todos los grupos de la entrada
```

```
ARRAY TEXT($_tabAttributes;0)
ARRAY BOOLEAN($_tabAttributes_asArray;0)
APPEND TO ARRAY($_tabAttributes;"cn")
APPEND TO ARRAY($_tabAttributes_asArray;False)
APPEND TO ARRAY($_tabAttributes;"memberOf")
APPEND TO ARRAY($_tabAttributes_asArray;False)

LDAP LOGIN($url;$login;$pwd;LDAP password plain text)
LDAP SEARCH ALL($dnSearchRootEntry;$_entry;$filter;LDAP all levels;$_tabAttributes;$_tabAttributes_asArray)
LDAP LOGOUT

$memberOf:=OB Get($_entry{1};"memberOf")
 // en $memberOf tenemos una variable que contiene el primer grupo de la entrada
```

Lenguaje

- Command name
- Count parameters
- Current method name
- EXECUTE METHOD
- Get action info
- Get pointer
- INVOKE ACTION
- Is a variable
- Is nil pointer
- NO TRACE
- Null
- RESOLVE POINTER
- Self
- This New 17.0
- TRACE
- Type
- Undefined
- Value type

Command name

Command name (comando {; info {; theme}}) -> Resultado

Parámetro	Tipo		Descripción
comando	Entero largo	➡	Número del comando
info	Entero largo	⬅	Propiedad hilo seguro del comando
theme	Texto	⬅	Tema del lenguaje del comando
Resultado	Cadena	➲	Nombre del comando traducido

Descripción

El comando ***Command name*** *devuelve el nombre y (opcionalmente) las propiedades del comando cuyo número se pasa en comando.*

Nota: *el número de cada comando está indicado en el Explorador, así como también en el área de Propiedades de esta documentación.*

Nota de compatibilidad*: como el nombre de un comando puede variar de una versión 4D a otra (comandos renombrados), este comando se utilizó en versiones anteriores del programa para designar un comando directamente por medio de su número, especialmente en partes de código no tokenizadas. Esta sintaxis permite evitar posibles problemas debido a las variaciones en los nombres de los comandos, así como otros elementos tales como tablas, sin dejar de poder escribir estos nombres de forma legible (para obtener más información sobre este punto, consulte la sección* ***Utilizar tokens en fórmulas****). Además, de forma predeterminada, la versión en Inglés del lenguaje se utiliza a partir de 4D v15; Sin embargo, la opción "Utilizar la configuración del sistema regional" en la* ***Página Métodos*** *de las Preferencias le permite continuar con el uso de la versión en español en un 4D en español.*

Dos parámetros opcionales están disponibles:

- *info: propiedades del comando. El valor devuelto es un campo de bits, donde actualmente sólo el primer bit (bit 0) es significativo. Se pone en 1 si el comando es hilo seguro (es decir, compatible con la ejecución de un proceso apropiativo) y 0 si es hilo-inseguro. Sólo los comandos compatibles con el proceso se pueden utilizar en los procesos apropiativos. Para más información sobre este punto, consulte la sección* ***Procesos 4D apropiativos****.*
- *tema: devuelve el nombre del tema del comando en el lenguaje 4D.*

El comando ***Command name*** *define la variable OK en 1 si comando corresponde a un número de comando existente y a 0 en caso contrario. Tenga en cuenta, sin embargo, que algunos comandos existentes han sido desactivados, en cuyo caso* ***Command name*** *devuelve una cadena vacía (ver el último ejemplo).*

Ejemplo 1

El siguiente código le permite cargar todos los comandos 4D válidos en un array:

```
C_LONGINT($Lon_id)
C_TEXT($Txt_command)
ARRAY LONGINT($tLon_Command_IDs;0)
ARRAY TEXT($tTxt_commands;0)

Repeat
   $Lon_id:=$Lon_id+1
   $Txt_command:=Command name($Lon_id)
   If(OK=1) //el número de comando existe
      If(Length($Txt_command)>0) //el comando no está desactivado
         APPEND TO ARRAY($tTxt_commands;$Txt_command)
         APPEND TO ARRAY($tLon_Command_IDs;$Lon_id)
      End if
   End if
Until(OK=0) //fin de los comandos existentes
```

Ejemplo 2

En un formulario, usted quiere mostrar una lista desplegable que contenga los comandos estándar de generación de informes. En el método de objeto de esta lista desplegable, usted escribe:

```
Case of
   :(Form event=On Before)
      ARRAY TEXT(asCommand;4)
      asCommand{1}:=Command name(1)
      asCommand{2}:=Command name(2)
      asCommand{3}:=Command name(4)
      asCommand{4}:=Command name(3)
   ` ...
End case
```

En la versión inglesa de 4D, la lista desplegable contendrá: Sum, Average, Min, y Max. En la versión francesa de 4D, la lista desplegable contendrá: Somme, Moyenne, Min, y Max.*

**con la aplicación 4D configurada para ser utilizada en idioma francés (ver nota de compatibilidad).*

Ejemplo 3

*Desea crear un método que devuelva **True** si el comando, cuyo número se pasa como parámetro, es hilo seguro y en caso contrario **False**.*

```
//Método proyecto Is_Thread_Safe
//Is_Thread_Safe(numCom) -> Booleano

C_LONGINT($1;$threadsafe)
C_TEXT($name)
C_BOOLEAN($0)
$name:=Command name($1;$threadsafe;$theme)
If($threadsafe ?? 0) //si el primer bit se define en 1
   $0:=True
Else
   $0:=False
End if
```

Luego, para el comando "SAVE RECORD" (53) por ejemplo, puede escribir:

```
$isSafe:=Is_Thread_Safe(53)
// devuelveTrue
```

Count parameters

Count parameters -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo	➲	Número de parámetros efectivamente pasados

Descripción

*El comando **Count parameters** devuelve el número de parámetros pasados a un método de proyecto.*

***Advertencia: Count parameters** es significativo sólo en un método de proyecto que haya sido llamado por otro método (método de proyecto u otro). Si el método de proyecto que llama **Count parameters** está asociado a un menú, **Count parameters** devuelve 0.*

Ejemplo 1

Los métodos de proyecto de 4D aceptan parámetros opcionales, a partir de la derecha.
*Por ejemplo, puede llamar al método **MiMetodo(a;b;c;d)** de las siguientes formas:*

```
MiMetodo(a;b;c;d)  ` Todos los parámetros se pasan
MiMetodo(a;b;c)  ` El último parámetro no se pasa
MiMetodo(a;b)  ` Los dos últimos parámetros no se pasan
MiMetodo(a)  ` Sólo se pasa el primer parámetro
MiMetodo  ` No se pasa ningún parámetro
```

*Utilizando **Count parameters** desde **MiMetodo,** puede detectar el número de parámetros pasados y efectuar diferentes operaciones dependiendo de lo que haya recibido. El siguiente ejemplo muestra un mensaje de texto y puede insertar el texto en un área 4D Write o enviar el texto a un documento en disco:*

```
  ` Método de proyecto AGREGAR TEXTO
  ` AGREGAR TEXTO ( Texto { ; Entero largo { ; Hora } } )
  ` AGREGAR TEXT ( Texto { ; Área 4D Write { ; RefDoc } } )

C_TEXT($1)
C_TIME($2)
C_LONGINT($3)

MESSAGE($1)
If(Count parameters>=3)
   SEND PACKET($3;$1)
Else
   If(Count parameters>=2)
      WR INSERT TEXT($2;$1)
   End if
End if
```

Después de añadir este método de proyecto a su aplicación, puede escribir:

```
AGREGAR TEXTO(vtText)  ` Mostrar sólo el mensaje de texto
AGREGAR TEXTO(vtText;$wrArea)  ` Mostrar el mensaje de texto y añadir el texto a $wrArea
AGREGAR TEXTO(vtText;0;$vhRefDoc)  ` Mostrar el mensaje de texto y escribirlo en $vhDocRef
```

Ejemplo 2

*Los métodos de proyecto de 4D aceptan un número variable de parámetros del mismo tipo, a partir de la derecha. Para declarar estos parámetros, utilice las directivas de compilación a las cuales usted pasa ${N} como una variable, donde **N** especifica el primer parámetro. Utilizando **Count parameters** puede referenciar estos parámetros con un bucle For y la sintaxis de indirección de parámetro. Este ejemplo es una función que devuelve el número más grande recibido como parámetro:*

```
  ` Método de proyecto Max de
  ` Max de ( Real { ; Real2... ; RealN } ) -> Real
  ` Max de ( Valor { ; Valor2... ; ValorN } ) -> Valor máximo

C_REAL($0;${1})  ` Todos los parámetros son de tipo REAL así como el resultado de la función
$0:=${1}
For($vlParam;2;Count parameters)
   If($${$vlParam}>$0)
      $0:=${$vlParam}
```

```
    End if
End for
```

Después de añadir este método de proyecto a su aplicación, puede escribir:

```
vrResult:=Max of(Records in set("Operación A");Records in set("Operación B"))
```

o:

```
vrResult:=Max of(r1;r2;r3;r4;r5;r6)
```

Current method name

Current method name -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Cadena*		*Nombre del método de llamada*

Descripción

*El comando **Current method name** devuelve el nombre del método en el cual se llama. Este comando es útil para depuración de métodos genéricos.*

En función del tipo de método llamado, la cadena devuelta puede ser:

Método de llamada	*Cadena devuelta*
Método base	*NomMetod*
Trigger	*Trigger on [NomTabla]*
Método proyecto	*NomMetod*
Método formulario tabla	*[NomTabla].NomFormulario*
Método formulario proyecto	*NomFormulario*
Método objeto formulario tabla	*[NomTabla].NomFormulario.NomObjeto*
Método objeto formulario proyecto	*NomFormulario.NomObjeto*
Método proyecto de componente	*NomMetodo*
Método formulario proyecto de componente	*NomFormulario (NomComponente)*
Método objeto formulario proyecto de componente	*NomFormulario (NomComponente).NomObjeto (NomComponente)*

Este comando no puede llamarse desde una fórmula 4D.

***Nota:** para que este comando funcione en modo compilado, la base debe haber sido compilada con la opción **Control de ejecución** (ubicada en las Preferencias de la aplicación) activada.*
Para desactivar localmente el control de ejecución en un método (o en una parte de un método), puede utilizar los siguientes comentarios especiales:

```
`%R- para desactivar el control de ejecución
`%R+ para activar el control de ejecución
`%R* para restituir el estado inicial de control de ejecución (definido en las Preferencias).
```

EXECUTE METHOD

EXECUTE METHOD (nomMetodo {; result {; param}}{; param2 ; ... ; paramN})

Parámetro	Tipo		Descripción
nomMetodo	Cadena	➡	Nombre del método de proyecto a ejecutar
result	Variable, Operador	⬅	Variable que recibe el resultado del método * para un método que no devuelve un resultado
param	Expresión	➡	Parámetro(s) del método

Descripción

El comando **EXECUTE METHOD** provoca la ejecución del método de proyecto nomMetodo pasando los parámetros en param1...paramN. Puede pasar el nombre de un método que puede ser llamado desde la base o el componente que ejecuta el comando.

En result, puede pasar una variable que reciba el resultado de la ejecución de nomMetodo (valor ubicado en $0 al interior de nomMetodo). Si el método no devuelve un resultado, pase * como segundo parámetro.

El contexto de ejecución se conserva en el método llamado, lo que significa que el formulario actual y el evento de formulario actual permanecen definidos.

Si llama este comando desde un componente y pasa un nombre de método que pertenece a la base local en nomMetodo (o viceversa), el método debe estar compartido (opción "Compartido entre componente y base principal", en las propiedades del método).

Variables y conjuntos del sistema

Si este comando se ejecuta correctamente, la variable sistema OK toma el valor 1; de lo contrario toma el valor 0.

Get action info

Get action info (action {; objetivo}) -> Resultado

Parámetro	Tipo		Descripción
action	Cadena	➡	Nombre o patrón de acción estándar incluyendo el parámetro si es necesario
objetivo	Entero largo	➡	Define el objetivo de la acción para obtener información: forma principal o forma actual
Resultado	Objeto	⬅	Objeto que contiene el estado de la acción como propiedades booleanas: isEnabled, isVisible, isChecked, isMixed, isUnknownState

Descripción

*El comando **Get action info** devuelve varias informaciones, incluida la disponibilidad y el estado, sobre la accion definida en el objetivo, de acuerdo con el contexto de la aplicación actual.*

*En accion, pase el nombre de la acción estándar a verificar. Puede ser una cadena o una constante del tema **Acción estándar**. La lista detallada de acciones se ofrece en la sección **Acciones estándar** del manual de Diseño 4D.*

***Nota:** algunas acciones aceptan parámetros. En este caso, debe utilizar el patrón siguiente: actionName?parameterName=parameterValue. Ejemplo: "gotoPage?value=2"*

*Puede pasar en objetivo el contexto del formulario en el que se debe ejecutar la accion, si está disponible. Puede utilizar una de las siguientes constantes del tema **Acción estándar**:*

Constante	Tipo	Valor	Comentario
ak current form	Entero largo	1	El formulario actual es el formulario donde se llamó la acción. Podría ser el formulario principal o un formulario tipo paleta delante del formulario principal del proceso actual.
ak main form	Entero largo	2	El formulario principal es el documento más adelante o el formulario diálogo del proceso, excluyendo cualquier ventana flotante o emergente.

***Nota:** si se omite el objetivo, se utiliza por defecto el contexto ak current form.*

*El comando **Get action info** devuelve información en forma de un objeto que contiene las siguientes propiedades:*

Propiedad	Tipo	Descripción
activado	Booleano	true si se puede invocar la acción, false en caso contrario
estado	Cadena	El valor puede ser una de las siguientes cadenas: "seleccionada": la acción está seleccionada, lo que significa que la propiedad está definida. Ejemplo: el texto seleccionado está en negrita, la propiedad "estado" de la acción estándar ak font bold contiene "seleccionado" "no seleccionado": la acción estándar no está seleccionada, lo que significa que la propiedad no está definida. Ejemplo: el texto seleccionado no está en negrita, la propiedad "estado" de la acción estándar ak font bold contiene "no seleccionado". "combinada": la acción es combinada, lo que significa que la propiedad está parcialmente definida. Ejemplo: arte del texto seleccionado está en "negrita", la propiedad "estado" de la acción estándar ak font bold contiene "combinada".
título	Text	Nombre actual localizado de la etiqueta de acción. Ejemplo: "Deshacer, "Pegar", etc. para la versión en inglés.
visible	Booleano	true si la acción es visible en el formulario
valor	Cadena	Valor actual de la cadena de parámetros de acción (si existe). Por ejemplo, si la acción estándar es "fontSize?value=10pt", la propiedad valor contiene "10pt"

Si no se puede determinar el estado de la acción (por ejemplo, si no se afecta a ningún objeto o comando de menú), el comando devuelve un objeto nulo (indefinido).

Ejemplo

Desea saber si la acción copiar está disponible (es decir, si se han seleccionado algunos datos):

```
C_OBJECT($actionInfo)
C_BOOLEAN($isEnabled)
$actionInfo:=Get action info(ak copy)
If(OB Is defined($actionInfo)) //la acción es definida en el proceso
   If(OB Get($actionInfo;"enabled"))
  //la acción copiar está disponible
   End if
End if
```

Get pointer

Get pointer (nomVar) -> Resultado

Parámetro	Tipo		Descripción
nomVar	Cadena	➡	Nombre de una variable proceso o interproceso
Resultado	Puntero	➲	Puntero hacia una variable proceso o interproceso

Descripción

*El comando **Get pointer** devuelve un puntero hacia una variable proceso o interproceso cuyo nombre se pasa en nomVar.*
*Para obtener un puntero hacia un campo, utilice **Field**. Para obtener un puntero hacia una tabla, utilice **Table**.*
***Nota:** puede pasar a **Get pointer** expresiones como por ejemplo, nomArray+"{3}", así como también elementos de array 2D (nomArray+"{3}{5}").*
Sin embargo, puede pasar elementos de variables (nomArray+"{myVar}").

Ejemplo 1

En un formulario, usted construye una matriz de 5 x 10 de variables editables llamadas v1, v2... v50. Para inicializar todas estas variables, usted escribe:

```
` ...
For($vlVar;1;50)
   $vpVar:=Get pointer("v"+String($vlVar))
   $vpVar->:=""
End for
```

Ejemplo 2

Utilizar punteros a elementos de arrays de dos dimensiones:

```
$pt:=Get pointer("a{1}{2}")
 //$pt=->a{1}{2}
$pt2:=Get pointer("atCities"+"{2}{6}")
 //$pt2=->atCities{2}{6}
```

INVOKE ACTION

INVOKE ACTION (accion {; objetivo})

Parámetro	Tipo		Descripción
accion	Cadena	➡	Nombre o patrón de acción estándar incluyendo parámetro si es necesario
objetivo	Entero largo	➡	Define dónde ejecutar la acción: formulario actual (por defecto) o formulario principal

Descripción

El comando **INVOKE ACTION** activa la acción estándar definida por el parámetro accion, opcionalmente en el contexto objetivo.
En accion, pase el nombre de la acción estándar a ejecutar. Puede ser una cadena o una constante del tema **Acción estándar**.
Todas las acciones disponibles se listan en la sección **Acciones estándar** del manual de Diseño 4D.

Notas:

- Algunas acciones aceptan un parámetro. En este caso, debe utilizar el patrón siguiente: actionName?parameterName=parameterValue. Ejemplo: "gotoPage?value=2"
- También se ofrecen acciones específicas adicionales para los documentos 4D Write Pro. Se detallan en la sección **Uso de acciones estándar** del manual 4D Write Pro Reference.

En objetivo, puede pasar el contexto del formulario en el que se debe ejecutar la accion. Puede utilizar una de las siguientes constantes del tema **Acción estándar**:

Constante	Tipo	Valor	Comentario
ak current form	Entero largo	1	El formulario actual es el formulario donde se llamó la acción. Podría ser el formulario principal o un formulario tipo paleta delante del formulario principal del proceso actual.
ak main form	Entero largo	2	El formulario principal es el documento más adelante o el formulario diálogo del proceso, excluyendo cualquier ventana flotante o emergente.

Nota: si se omite objetivo, se utiliza por defecto el contexto ak current form.
Dependiendo del objetivo, la ejecución del comando **INVOKE ACTION** es síncrona o asíncrona:

- Con ak current form como objetivo, el comando **INVOKE ACTION** es síncrono; La acción se ejecuta en el ciclo actual en el momento en que se llama al comando.
- Con ak main form como objetivo, el comando **INVOKE ACTION** es asíncrono; La acción se ejecuta en el siguiente ciclo después del final de la ejecución del método objeto de formulario.

Nota: las acciones de edición estándar (Cortar, Copiar, Pegar, Seleccionar todo, Borrar, Deshacer/Rehacer) ignoran el parámetro objetivo, si se pasa. Tales acciones se ejecutan siempre de forma sincrónica en el contexto del objeto editable que tiene el foco.
El comando **INVOKE ACTION** no genera un error, por ejemplo, si la acción solicitada no está disponible en el contexto actual. Debe validar la acción esperada utilizando el comando **Get action info**.

Ejemplo 1

Desea ejecutar la acción estándar **Copiar** en el formulario actual:

```
INVOKE ACTION(ak copy;ak current form)
```

Ejemplo 2

Usted desea ejecutar una acción estándar **Goto page** (página 3) en el formulario principal:

```
INVOKE ACTION(ak goto page+"?value=3";ak main form)
```

Is a variable

Is a variable (puntero) -> Resultado

Parámetro	*Tipo*		*Descripción*
puntero	*Puntero*	➡	*Puntero a probar*
Resultado	*Booleano*	⮌	*TRUE = Puntero apunta a una variable FALSE = Puntero no apunta a una variable*

Descripción

*El comando **Is a variable** devuelve True si el puntero pasado en unPuntero referencia a una variable definida. Devuelve False en todos los otros casos (puntero hacia un campo o tabla, puntero Nil, etc.).*

*Si quiere conocer el nombre de la variable que está siendo apuntada o el número del campo, puede utilizar el comando **RESOLVE POINTER**.*

Is nil pointer

Is nil pointer (puntero) -> Resultado

Parámetro	Tipo		Descripción
puntero	Puntero	➡	Puntero a probar
Resultado	Booleano	⮌	TRUE = Puntero Nil (->[]) FALSE = Puntero válido hacia un objeto existente

Descripción

*El comando **Is nil pointer** devuelve True si el puntero que pasa en unPuntero es Nil (->[]). Devuelve False en todos los otros casos (puntero hacia un campo, tabla o variable).*
*Si quiere conocer el nombre de la variable apuntada o el número del campo, puede utilizar el comando **RESOLVE POINTER**.*

Ejemplo

```
C_POINTER($ptr)
...
If(Is nil pointer($ptr))
End if
 // is equivalent to
If($ptr=Null)
End if
```

NO TRACE

NO TRACE
Este comando no requiere parámetros

Descripción

*El comando **NO TRACE** se utiliza durante el desarrollo de una base de datos, para controlar la ejecución de los métodos.*
***NO TRACE** desactiva el depurador llamado por **TRACE**, por un error o por el usuario. Utilizar **NO TRACE** tiene el mismo efecto que hacer clic en el botón Reanudar en el depurador.*
*En bases compiladas, se ignora el comando **NO TRACE**.*

Null

Null -> Resultado

Parámetro	Tipo		Descripción
Resultado	Null	➲	Valor Null

Descripción

Null devuelve el valor **null** de tipo null.

Esta función permite afectar o comparar el valor **null** de los siguientes elementos del lenguaje 4D:

Elementos del lenguaje	Comentarios
Valores de propiedades de objetos	La comparación de **Null** con una propiedad de objeto devuelve True si el valor de la propiedad es null o False de lo contrario. Para simplificar código, comparar **Null** también devuelve true si la propiedad no existe en el objeto (es decir **Undefined**), ver ejemplo 4.
Elementos de colecciones	Cuando una colección se expande añadiendo elementos no adyacentes, todos los elementos intermediarios obtienen automáticamente el valor **null**.
Variables de tipo objeto (**C_OBJECT**)	Ver (*) abajo
Variables de tipo colección (**C_COLLECTION**)	Ver (*) abajo
Variables de tipo puntero (**C_POINTER**)	Ver (*) abajo
Variables de tipo imagen (**C_PICTURE**)	(*) Asignar el valor null a una variable de este tipo borra su contenido. En este caso, tiene el mismo efecto que llamar al comando **CLEAR VARIABLE**.

El valor **Null** no se puede pasar como un parámetro a un método o se devuelve como un resultado de función.

Nota: este comando no se puede utilizar con campos escalares de la base de datos. Los valores Null en los campos de la base son gestionados por el motor SQL y se gestionan a través de los comandos **Is field value Null** y **SET FIELD VALUE NULL**.

Ejemplo 1

Usted desea asignar y probar el valor **null** con las propiedades de los objetos:

```
C_OBJECT(vEmp)
vEmp:=New object
vEmp.name:="Smith"
vEmp.children:=Null

If(vEmp.children=Null) //true
End if
If(vEmp.name=Null) //false
End if
If(vEmp.parent=Null) //true
End if
```

Nota: este ejemplo requiere que la notación objeto esté activada en la base.

Ejemplo 2

Usted desea asignar y comparar el valor **null** con colección de elementos:

```
C_COLLECTION(myCol)
myCol:=New collection(10;20;Null)
...
If(myCol[2]=Null)
  // si el tercer elemento es null
    ...
End if
```

Ejemplo 3

Estos ejemplos muestran las distintas maneras de asignar o comparar el valor **null** con las variables:

```
 //Variable objeto
C_OBJECT($o)
$o:=New object
$o:=Null  //equivalente a CLEAR VARIABLE($o)
If($o#Null)  //equivalente a If (OB Is defined($o))
End if
```

```
 //Variable collection
C_COLLECTION($c)
$c:=New collection
$c:=Null  //equivalent to CLEAR VARIABLE($c)
If($c#Null)
End if
```

```
 //Variable Puntero
C_POINTER($p)
$p:=->$v
$p:=Null  //equivalente a CLEAR VARIABLE($p)
If($p=Null)  //equivalente a If (Is Nil pointer($p))
End if
```

```
 //Variable imagen
C_PICTURE($i)
$i:=$vpicture
$i:=Null  //equivalente a CLEAR VARIABLE($i)
If($i#Null)  //equivalente a If (Picture size($i)#0)
End if
```

Ejemplo 4

*Aquí están los diferentes resultados del comando **Undefined** así como también del comando **Null** aplicados a las propiedades de objetos, dependiendo del contexto:*

```
C_OBJECT(vEmp)
vEmp:=New object
vEmp.name:="Smith"
vEmp.children:=Null

$undefined:=Undefined(vEmp.name)  // False
$null:=(vEmp.name=Null)  //False

$undefined:=Undefined(vEmp.children)  // False
$null:=(vEmp.children=Null)  //True

$undefined:=Undefined(vEmp.parent)  // True
$null:=(vEmp.parent=Null)  //True
```

RESOLVE POINTER

RESOLVE POINTER (puntero ; nomVar ; numTabla ; numCamp)

Parámetro	Tipo		Descripción
puntero	Puntero	➡	Puntero del cual recuperar el objeto referenciado
nomVar	Cadena	⬅	Nombre de la variable referenciada o cadena vacía
numTabla	Entero largo	⬅	Número de la tabla o del elemento del array referenciado o 0 o -1
numCamp	Entero largo	⬅	Número del campo referenciado o 0

Descripción

*El comando **RESOLVE POINTER** recupera la información del objeto referenciado por la expresión de puntero puntero y la devuelve en los parámetros nomVar, numTabla, y numCamp.*

*Dependiendo de la naturaleza del objeto referenciado, **RESOLVE POINTER** devuelve los siguientes valores:*

Objeto referenciado	Parámetros			
	nomVar	numTabla	numCamp	
Nada (NIL pointer)	"" (cadena vacía)	0		0
Variable	Nombre de la variable	-1		-1
Array	Nombre del array	-1		0
Elemento de array	Nombre del array	número del elemento		-1
Elemento de array 2D	Nombre del array 2D	número de la fila del elemento	número de la columna del elemento	
Tabla	"" (cadena vacía)	número de la tabla		0
Campo	"" (cadena vacía)	número de la tabla		número del campo

Notas:

• *Si el valor que pasa en puntero no es una expresión de tipo puntero, se generará un error de sintaxis.*
• *El comando **RESOLVE POINTER** no funciona con punteros a variables locales. De hecho, por definición muchas variables locales con el mismo nombre podrían existir en diferentes lugares, de manera que no es posible para el comando encontrar la variable correcta.*

Ejemplo 1

En un formulario, usted crea un grupo de 100 variables editables llamadas v1, v2... v100. Para hacer esto, usted realiza los siguientes pasos:

a. Crea una variable editable que llama v.

b. Define las propiedades del objeto.

c. Asocia el siguiente método al objeto:

```
HacerAlgo(Self) ` HacerAlgo es un método de proyecto de su base
```

d. En este punto, puede duplicar la variable tantas veces como sea necesario, o utilizar la funcionalidad Duplicar sobre matriz en el editor de formularios.

*e. En el método **HacerAlgo**, si necesita conocer el índice de la variable para la cual se llama el método, escribe:*

```
RESOLVE POINTER($1;$vsNomVar;$vlNumTabla;$vlNumCampo)
$vlNomVar:=Num(Substring($vsNomVar;2))
```

*Note que construyendo su formulario de esta manera, usted escribe los métodos para las 100 variables sólo una vez; no necesita escribir **HacerAlgo (1), HacerAlgo (2)...,HacerAlgo (100).***

Ejemplo 2

Por propósitos de depuración, necesita verificar que el segundo parámetro ($2) de un método es un puntero a una tabla. Al comienzo de este método, escribe:

```
` ...
If(◇DebugOn)
   RESOLVE POINTER($2;$vsNomVar;$vlNumTabla;$vlNumCampo)
   If(Not(($vlNumTabla>0)&($vlNumCampo=0)&($vsNomVar="")))
` ATENCIÓN: El puntero no es una referencia a una tabla
      TRACE
      End
```

```
   End if
` ...
```

Ejemplo 3

*Ver el ejemplo del comando **DRAG AND DROP PROPERTIES**.*

Ejemplo 4

Este es un ejemplo de puntero a un array 2D:

```
ARRAY TEXT(atCities;100;50)
C_POINTER($city)
atCities{1}{2}:="Rome"
atCities{1}{5}:="Paris"
atCities{2}{6}:="New York"
 // ...otros valores
$city:=->atCities{1}{5}
RESOLVE POINTER($city;$var;$rowNum;$colNum)
 //$var="atCities"
 //$rowNum="1"
 //$colNum="5"
```

Self

Self -> Resultado

Parámetro	Tipo	Descripción
Resultado	Puntero	Puntero hacia el objeto de formulario (si lo hay) cuyo método está siendo ejecutado actualmente. Si no Nil (->[]) si fuera de contexto

Nota de compatibilidad

*Este comando sólo se conserva por razones de compatibilidad. A partir de la versión 12 de 4D, se recomienda utilizar el comando **OBJECT Get pointer**.*

Descripción

*El comando **Self** devuelve un puntero hacia el objeto cuyo método de objeto se está ejecutando.*

***Self** se utiliza para referenciar una variable en su propio método de objeto. Devuelve un puntero válido sólo cuando se desde dentro de un método de objeto o desde un método de proyecto que se llama directa o indirectamente por un método de objeto.*

*Si **Self** se llama fuera de contexto, devuelve un puntero Nil (->[]).*

***Consejo: Self** es muy útil cuando varios objetos en un formulario deben efectuar la misma acción, operada sobre ellos mismos.*

***Nota:** cuando se utiliza en el contexto de un list box, la función devuelve:*

- *Para una columna asociada a un campo, un puntero al campo asociado,*
- *Para una columna asociada a una variable, un puntero a la variable,*
- *Para una columna asociada a una expresión, un puntero Nil.*

Ejemplo

*Ver el ejemplo del comando **RESOLVE POINTER**.*

This

This -> Resultado

Parámetro	Tipo	Descripción
Resultado	Objeto	El elemento actual

Descripción

*El comando **This** devuelve una referencia al objeto procesado actualmente.*
El comando está diseñado para usarse en el siguiente contexto:

- *un list box asociado a una colección o una selección de entidades (entity selection),*
- *durante el evento formulario On Display Detail o el evento formulario On Data Change.*

*En este contexto, el comando devuelve una referencia al elemento de colección o la entidad a la cual el list box accede para mostrar la línea actual. En cualquier otro contexto, el comando devuelve **Null**.*

*Puede acceder a todas las propiedades de los elementos o todos los atributos de entidades vía **This.<propertyPath>**. Por ejemplo, This.name o This.employer.lastName son rutas de propiedades de elementos o de entidades (atributos) validos.*

***Nota:** si utiliza una colección de valores escalares, 4D crea un objeto para cada elemento con una sola propiedad **valor**. Por lo tanto, el valor del elemento está disponible a través de la expresión **This.value**.*

Ejemplo 1

Una colección de objetos, cada uno con esta estructura:

{ "ID": 1234 "name": "Xavier", "revenues": 47300, "employees": ["Allan", "Bob", "Charlie"] },{ "ID": 2563 "name": "Carla", "revenues": 55000, "isFemale": true "employees": ["Igor", "Jane"] },...

En el list box, cada columna se refiere a una de las propiedades del objeto, ya sea directamente (This.name), indirectamente (This.employees.length), o mediante una expresión (getPicture) en la que se puede usar directamente. El list box se ve así:

ID	Name	Revenues	Employee count	Picture
This.ID	This.name	This.revenues	This.employees.length	getPicture

*El método proyecto GetPicture se ejecuta automáticamente durante el evento **On display detail**:*

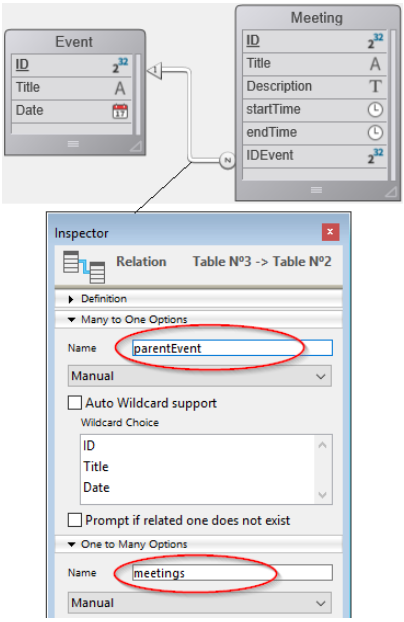
```
//Método GetPicture
C_PICTURE($0)
If(This.isFemale)
    $0:=Form.genericFemaleImage
Else
    $0:=Form.genericMaleImage
End if
```

Una vez el formulario se ejecuta, puede ver el resultado:

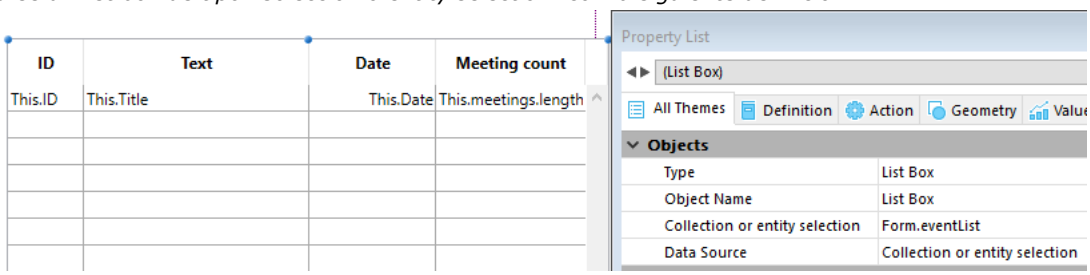
ID	Name	Revenues	Employee count	Picture
1234	Xavier	47300	3	
452	Henry	54000	5	
2563	Carla	55000	2	
65	Mike	66000	4	
832	Vanessa	42010	6	
57	Mary	46000	5	

Ejemplo 2

Usted quiere mostrar las siguientes estructuras en un list box:



Cree un list box de tipo "Colección o entity selection" con la siguiente definición:



Tenga en cuenta que:

- *This.ID, This.Title y This.Date directly se refieren directamente a los atributos correspondientes en la clase de datos ds.Event.*
- *This.meetings es un atributo relacional (basado en el nombre de relación Unos A Muchos) que devuelve una selección de entidades de la clase de datos ds.Meeting.*
- ***Form.eventList** es la selección de entidades que se asocia al list box. El código de inicialización se puede poner en el evento formulario On load:*

```
Case of
   :(Form event=On Load)
      Form.eventList:=ds.Event.all() //devuelve una selección de entidades con todas las entidades
End case
```

Una vez se ejecuta el formulario, el list box se llena automáticamente con la selección de entidades:

ID	Text	Date	Meeting count
1	Bliss Feast	12/01/2018	3
2	Remembrance Feast	24/10/2018	1
3	Heroes Feast	03/06/2018	2
4	Meat Fest	15/09/2018	1
5	Day of Ores	28/03/2018	3

TRACE

TRACE
Este comando no requiere parámetros

Descripción

*El comando **TRACE** se utiliza para ejecutar paso a paso métodos durante el desarrollo de una base.*

*El comando **TRACE** muestra el depurador de 4D en el proceso actual. La ventana del depurador aparece antes de la ejecución de la línea de código siguiente, y continúa para cada línea de código que se ejecuta. Igualmente puede llamar manualmente al depurador presionando **Alt+Mayús+clic derecho** (Windows) o **Control+Opción+comando+clic** (Macintosh) durante la ejecución del código.*

*En bases de datos compiladas, el comando **TRACE** se ignora.*

***4D Server:** si llama **TRACE** desde un método de proyecto ejecutado en el contexto de un Procedimiento almacenado, la ventana del depurador aparece en el equipo servidor.*

***Consejo:** no llame **TRACE** cuando utilice un formulario para el cual los eventos **On Activate** y **On Deactivate** hayan sido activados. Cada vez que la ventana del depurador aparezca, estos eventos serán invocados; esto creará un bucle infinito entre estos eventos y la ventana del depurador. Si termina en esta situación, utilice la combinación **Mayús+clic** en el botón **Reanudar** del depurador para salir de ahí. Cualquier llamada posterior a **TRACE** dentro del proceso será ignorada.*

Ejemplo

*El siguiente código espera que la variable proceso **CREAR_LENG** sea igual a "US" o "FR". Si no es el caso, llama al método de proyecto **DEBUG**:*

```
` ...
Case of
   :(CREAR_LENG="US")
      vsBHCmdNom:=[Comandos]CM US Nom
   :(CREAR_LENG="FR")
      vsBHCmdNom:=[Comandos]CM FR Nom
   Else
      DEBUG("Valor de CREAR_LENG")
End case
```

*El método de proyecto **DEBUG** se lista aquí:*

```
` Método de proyecto DEBUG
` DEBUG (Texto)
` DEBUG (Información opcional de depuración)

C_TEXT($1)

If(◇vbDebugOn) ` Variable interproceso definida en el Método On Startup
   If(Compiled application)
      If(Count parameters>=1)
         ALERT($1+Char(13)+"Llamar al diseñador al x911")
      End if
   Else
      TRACE
   End if
End if
```

Type

Type (campoVar) -> Resultado

Parámetro	Tipo		Descripción
campoVar	Campo, Variable	➡	Campo o variable a probar
Resultado	Entero largo	⮌	Número de tipo de datos

Descripción

*El comando **Type** devuelve un valor numérico que indica el tipo de campo o variable que pasa en el parámetro campoVar. 4D ofrece las siguientes constantes predefinidas que se encuentran en el tema **Tipos de campos y variables**:*

Constante	Tipo	Valor
Array 2D	Entero largo	13
Blob array	Entero largo	31
Boolean array	Entero largo	22
Date array	Entero largo	17
Integer array	Entero largo	15
Is alpha field	Entero largo	0
Is BLOB	Entero largo	30
Is Boolean	Entero largo	6
Is collection	Entero largo	42
Is date	Entero largo	4
Is float	Entero largo	35
Is integer	Entero largo	8
Is integer 64 bits	Entero largo	25
Is longint	Entero largo	9
Is null	Entero largo	255
Is object	Entero largo	38
Is picture	Entero largo	3
Is pointer	Entero largo	23
Is real	Entero largo	1
Is string var	Entero largo	24
Is subtable	Entero largo	7
Is text	Entero largo	2
Is time	Entero largo	11
Is undefined	Entero largo	5
LongInt array	Entero largo	16
Object array	Entero largo	39
Picture array	Entero largo	19
Pointer array	Entero largo	20
Real array	Entero largo	14
String array	Entero largo	21
Text array	Entero largo	18
Time array	Entero largo	32

*Puede aplicar la función **Type** a los campos, variables interproceso, variables proceso, variables locales y punteros desreferenciados para estos tipos de objetos. Puede aplicar **Type** a los parámetros ($1, $2 ... ${...}) de un método proyecto o a un resultado de una función ($0).*

***Nota:** puede aplicar la función **Type** a las expresiones escalares como propiedades de objeto (emp.name) o los elementos de colecciones (myColl[5]). Para ello, debe utilizar el comando **Value type**.*

Ejemplo 1

El siguiente método de proyecto borra una parte o la totalidad de los campos del registro actual de la tabla a la cual apunta el puntero pasado como parámetro. Hace esto sin borrar o cambiar el registro actual:

```
  ` Método de proyecto BORRAR REGISTRO
  ` BORRAR REGISTRO ( Puntero {; Entero largo } )
  ` BORRAR REGISTRO ( -> [Tabla] { ; Tipo de valores } )
C_POINTER($1)
C_LONGINT($2;$vlTipoVal)
If(Count parameters>=2)
   $vlTipoVal:=$2
Else
   $vlTipoVal:=0xFFFFFFFF
End if
```

```
For($vlCampo;1;Count fields($1))
   $vpCampo:=Field(Table($1);$vlCampo)
   $vlTipoCampo:=Type($vpCampo->)
   If($vlTipoVal??$vlTipoCampo )
      Case of
         :(($vlTipoCampo =ls alpha field)|($vlTipoCampo =ls text))
            $vpCampo->:=""
         :(($vlTipoCampo =ls real)|($vlTipoCampo=ls integer)|($vlTipoCampo =ls longint))
            $vpCampo->:=0
         :($vlTipoCampo =ls date)
            $vpCampo->:=!00/00/00!
         :($vlTipoCampo =ls time)
            $vpCampo->:=?00:00:00?
         :($vlTipoCampo =ls Boolean)
            $vpCampo->:=False
         :($vlTipoCampo =ls picture)
            C_PICTURE($vgImagenVacia)
            $vpCampo->:=$vgImagenVacia
         :($vlTipoCampo =ls subtable)
            Repeat
               ALL SUBRECORDS($vpCampo->)
               DELETE SUBRECORD($vpCampo->)
            Until(Records in subselection($vpCampo->)=0)
         :($vlTipoCampo =ls BLOB)
            SET BLOB SIZE($vpCampo->;0)
      End case
   End if
End for
```

Después de implementar este método de proyecto en su base de datos, puede escribir:

```
  ` Borrar todo el registro actual de la tabla [Cosas por hacer]
BORRAR REGISTRO(->[Cosas por hacer])
  ` Borrar los campos de tipo Texto, BLOB e Imagen del registro actual de la tabla [Cosas por hacer]
BORRAR REGISTRO(->[Cosas por hacer];0?+ls text?+ls BLOB?+ls picture)
  ` Borrar la totalidad del registro actual de la tabla [Cosas por hacer] excepto los campos Alfa
BORRAR REGISTRO(->[Cosas por hacer];-1?-ls alpha field)
```

Ejemplo 2

En algunos casos, por ejemplo cuando se escribe código genérico, puede necesitar saber si un array es un array estándar independiente o una "fila" de un array 2D. En este caso, puede utilizar el siguiente código:

```
ptrmiArr:=->miArr{6}  ` ¿Es miArr{6} la fila de un array 2D?
RESOLVE POINTER(ptrmiArr;varNombre;numTabla;numCamp)
If(varNombre#"")
   $ptr:=Get pointer(varNombre)
   $eltipo:=Type($ptr->)
  ` Si miArr{6} es una fila de un array 2D, $eltipo es igual a 13
End if
```

Ejemplo 3

*Ver ejemplo del comando **APPEND DATA TO PASTEBOARD**.*

Ejemplo 4

*Ver ejemplo del comando **DRAG AND DROP PROPERTIES**.*

Undefined

Undefined (expresion) -> Resultado

Parámetro	Tipo		Descripción
expresion	Variable, Expresión, Campo	➡	Variable a probar
Resultado	Booleano	⮌	True = Variable indefinida False = Variable definida

Descripción

Undefined *devuelve True si no se ha definido el parámetro expresion y False si se definió expresion.*

- *Una variable está definida si se creó vía una directiva de compilación o si se le asigna un valor. Está indefinida en todos los demás casos. Si la base ha sido compilada, la función* ***Undefined*** *devuelve False para todas las variables.*
- *An object property is undefined if it does not exist in the object.*

Nota: *la función* ***Undefined*** *siempre devuerlve False a todas las referencias de campo.*

Ejemplo 1

El siguiente código administra la creación de procesos cuando se selecciona un elemento de menú de un módulo particular de la base. Si el proceso ya existe, usted lo pasa al primer plano; si no existe, usted lo inicia. Para hacer esto, para cada módulo de la aplicación, usted mantiene una variable interproceso ◇PID_... que inicializa en el método de base ***On Startup****.*

Al desarrollar la base, usted añade nuevos módulos. En lugar de modificar el método de base ***On Startup*** *(para añadir la inicialización de la variable* ***PID_...*** *correspondiente) y luego reabrir la base para reinicializar todo cada vez que añade un módulo, utilice el comando* ***Undefined*** *para administrar rápidamente la adición del nuevo módulo:*

```
// Método de proyecto M_AÑADIR_CLIENTES

If(Undefined(◇PID_AÑADIR_CLIENTES)) //Tenga en cuenta las etapas de desarrollo intermedias
   C_LONGINT(◇PID_AÑADIR_CLIENTES)
   ◇PID_AÑADIR_CLIENTES:=0
End if

If(◇PID_AÑADIR_CLIENTES=0)
   ◇PID_AÑADIR_CLIENTES:=New process("P_AÑADIR_CLIENTES";64*1024;"P_AÑADIR_CLIENTES")
Else
   SHOW PROCESS(◇PID_AÑADIR_CLIENTES)
   BRING TO FRONT(◇PID_AÑADIR_CLIENTES)
End if
// Nota: P_AÑADIR_CLIENTES, el método de gestión de procesos,
// da a ◇PID_AÑADIR_CLIENTES el valor cero cuando termina.
```

Ejemplo 2

Aquí están los diferentes resultados del comando ***Undefined*** *así como también del comando* ***Null*** *aplicados a las propiedades de objetos, dependiendo del contexto:*

```
C_OBJECT(vEmp)
vEmp:=New object
vEmp.name:="Smith"
vEmp.children:=Null

$undefined:=Undefined(vEmp.name) // False
$null:=(vEmp.name=Null) //False

$undefined:=Undefined(vEmp.children) // False
$null:=(vEmp.children=Null) //True

$undefined:=Undefined(vEmp.parent) // True
$null:=(vEmp.parent=Null) //True
```

Value type

Value type (expresion) -> Resultado

Parámetro	Tipo		Descripción
expresion	Expresión	➡	Expresión cuyo valor resultante debe ser probado
Resultado	Entero largo	⮌	Número de tipo de dato

Descripción

*El comando **Value type** devuelve el tipo del valor resultante de la evaluación de la expresion que pasó como parámetro.*
*El comando devuelve un valor numérico que se puede comparar con una de las siguientes constantes del tema **Tipos de campos y variables**:*

Constante	Tipo	Valor
Is BLOB	Entero largo	30
Is Boolean	Entero largo	6
Is collection	Entero largo	42
Is date	Entero largo	4
Is longint	Entero largo	9
Is null	Entero largo	255
Is object	Entero largo	38
Is picture	Entero largo	3
Is pointer	Entero largo	23
Is real	Entero largo	1
Is text	Entero largo	2
Is time	Entero largo	11
Is undefined	Entero largo	5
Object array	Entero largo	39

Este comando está designado para devolver el tipo de una expresión escalar, por ejemplo el valor almacenado o devuelto por el parámetro expresion. En particular, se puede aplicar a las siguientes expresiones 4D:

- *Propiedades del objetos (emp.name),*
- *Elementos de la colección (myCol[5]).*

***Nota**: las propiedades numéricas de los objetos siempre se consideran valores reales:*

```
C_OBJECT($o)
$o:=New object("value";42)
$vType:=Value type($o.value) //$vType=Is real
```

***Value type** se puede aplicar a cualquier expresión 4D válida, incluyendo campos, variables y parámetros. En este caso, a diferencia del comando **Type**, **Value type** devuelve el tipo internal del valor resultante de la evaluación de la expresion y no su tipo declarado. Dado que el lenguaje 4D convierte algunos tipos de valores internamente, el resultado de **Value type** puede diferir del tipo declarado. Por ejemplo, 4D convierte internamente los valores de campo de tipo "**Entero 64 bits**". Esto da los siguientes resultados:*

```
$vType1:=Type([myTable]Long64field) //$vType=Is integer 64 bits
$vType2:=Value type([myTable]Long64field) //$vType=Is real (en modo interpretado)
```

Otras diferencias están relacionadas con arrays (la evaluación de una array devuelve el índice de elementos actual) y el modo compilado. En la tabla siguiente se enumeran estas diferencias:

Tipo declarado	Type resultado	Value type resultado (interpretado)	Value type resultado (compilado)	Comentario
ARRAY TEXT($t;1)	Text array	Is real	Is longint	$t contiene el índice del elemento actual, que es un número
Campo **Alfa**	Is alpha field	Is text	Is text	4D maneja internamente todas las cadenas como textos
Campo **Entero**	Is integer	Is real	Is longint	Por motivos de optimización, en modo interpretado todos los valores numéricos se consideran reales y...
Campo **Entero largo**	Is longint	Is real	Is longint	... en modo compilado, todos los valores enteros se consideran enteros largos (*)
Campo **Entero 64 bits**	Is integer 64 bits	Is real	Is longint	

(*) Si desea escribir una prueba para un valor de tipo numérico válido para los modos compilado e interpretado, puede considerar el uso de un código como:

```
If(Value type($myValue)=Is longint)|(Value type($myValue)=Is real)
```

Nota de compatibilidad: *a partir de 4D v16 R6, las fechas se almacenan en las propiedades objeto, ya sea con el tipo fecha o como texto en formato de fecha ISO. Para más información, consulte el selector Dates inside objects del comando* ***SET DATABASE PARAMETER****.*

Ejemplo 1

Usted desea manejar los varios tipos posibles de un valor propiedad objeto:

```
Case of
   :(Value type($o.value)=Is real)
  //handle a numeric value
   :(Value type($o.value)=Is text)
  //handle a text
   :(Value type($o.value)=Is object)
  //handle a sub-object
      ...
End case
```

Ejemplo 2

Desea obtener la suma de todos los valores numéricos en una colección:

```
C_COLLECTION($col)
C_REAL($sum)
$col:=New collection("Hello";20;"World2";15;50;Current date;True;10)
For($i;0;$col.length-1) //-1 since collections start at 0
   If(Value type($col[$i])=Is real)
      $sum:=$sum+$col[$i]
   End if
End for
ALERT(String($sum)) //95
```

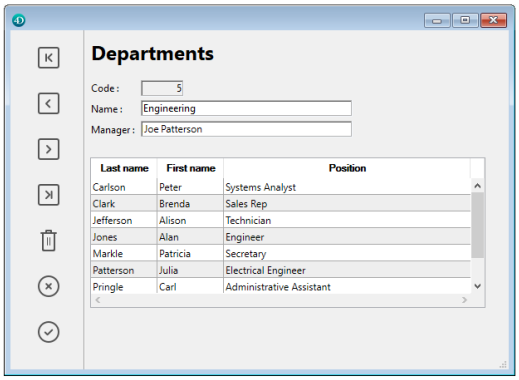
List Box

- *Gestión programada de los objetos de tipo List box*
- *Gestión de list box jerárquicos*
- *Utilizar arrays objetos en las columnas (4D View Pro)*
- *LISTBOX COLLAPSE*
- *LISTBOX DELETE COLUMN*
- *LISTBOX DELETE ROWS*
- *LISTBOX DUPLICATE COLUMN*
- *LISTBOX EXPAND*
- *LISTBOX Get array*
- *LISTBOX GET ARRAYS*
- *LISTBOX Get auto row height*
- *LISTBOX GET CELL COORDINATES*
- *LISTBOX GET CELL POSITION*
- *LISTBOX Get column formula*
- *LISTBOX Get column width*
- *LISTBOX Get footer calculation*
- *LISTBOX Get footers height*
- *LISTBOX GET GRID*
- *LISTBOX GET GRID COLORS*
- *LISTBOX Get headers height*
- *LISTBOX GET HIERARCHY*
- *LISTBOX Get locked columns*
- *LISTBOX Get number of columns*
- *LISTBOX Get number of rows*
- *LISTBOX GET OBJECTS*
- *LISTBOX GET PRINT INFORMATION*
- *LISTBOX Get property*
- *LISTBOX Get row color*
- *LISTBOX Get row font style*
- *LISTBOX Get row height*
- *LISTBOX Get rows height*
- *LISTBOX Get static columns*
- *LISTBOX GET TABLE SOURCE*
- *LISTBOX INSERT COLUMN*
- *LISTBOX INSERT COLUMN FORMULA* Updated 17.0
- *LISTBOX INSERT ROWS*
- *LISTBOX MOVE COLUMN*
- *LISTBOX MOVED COLUMN NUMBER*
- *LISTBOX MOVED ROW NUMBER*
- *LISTBOX SELECT BREAK*
- *LISTBOX SELECT ROW*
- *LISTBOX SET ARRAY*
- *LISTBOX SET AUTO ROW HEIGHT*
- *LISTBOX SET COLUMN FORMULA*
- *LISTBOX SET COLUMN WIDTH*
- *LISTBOX SET FOOTER CALCULATION*
- *LISTBOX SET FOOTERS HEIGHT*
- *LISTBOX SET GRID*
- *LISTBOX SET GRID COLOR*
- *LISTBOX SET HEADERS HEIGHT*
- *LISTBOX SET HIERARCHY*
- *LISTBOX SET LOCKED COLUMNS*
- *LISTBOX SET PROPERTY*
- *LISTBOX SET ROW COLOR*
- *LISTBOX SET ROW FONT STYLE*
- *LISTBOX SET ROW HEIGHT*
- *LISTBOX SET ROWS HEIGHT*
- *LISTBOX SET STATIC COLUMNS*
- *LISTBOX SET TABLE SOURCE*
- *LISTBOX SORT COLUMNS*

Gestión programada de los objetos de tipo List box

Los comandos de este tema están dedicados a la gestión de objetos de formulario de tipo List box.

Los list boxes permiten representar los datos en forma de columnas y de filas seleccionables y proponen numerosas funciones de interfaz como la posibilidad de introducir valores, ordenar columnas, mostrar una jerarquía, definir colores alternos, etc.



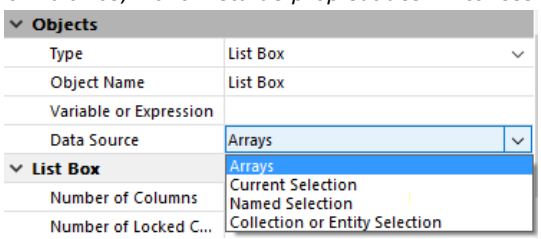
Es posible configurar completamente un list box en el editor de formularios de 4D e igualmente puede controlarlo por programación. Para mayor información sobre la creación y definición de objetos de tipo list box en el editor de formularios como también sobre su uso, consulte el Manual de Diseño de la documentación 4D.

La programación de objetos de tipo List box se efectúa de la misma forma que para los otros objetos de formulario en lista de 4D. Sin embargo, se deben seguir las reglas específicas, descritas en esta sección.

Fuentes de datos y principios de gestión de valores

Un list box puede contener una o más columnas y puede estar asociado a arrays 4D, a una selección de registros, a una colección o a una selección de entidades (entity selection). En el caso de los list box de tipo selección o colección/entity selection, las columnas son asociadas a las expresiones (las columnas de list box de tipo selección pueden igualmente asociarse a los campos).

No es posible combinar en un mismo list box tipos de fuentes de datos diferentes (arrays, selecciones, colecciones o selecciones de entidades). La definición de la fuente de datos se efectúa en el momento de la creación del list box en el editor de formularios, vía la Lista de propiedades. Entonces ya no es posible modificarlo por programación.



List box de tipo array

*En este tipo de list box, cada columna está asociada a un array 4D de una dimensión; todos los tipos de array pueden utilizarse, a excepción de los arrays de punteros. El formato de salida de cada columna puede definirse en el editor de formularios o utilizando el comando **OBJECT SET FORMAT**.*

*Utilizando el lenguaje, los valores de las columnas (entrada y visualización de datos) se administran utilizando los comandos de alto nivel del tema List box (tales como **LISTBOX INSERT ROWS** o **LISTBOX INSERT COLUMN**) como también los comandos de manipulación de arrays.*

Por ejemplo, para inicializar el contenido de una columna, puede utilizar la siguiente instrucción:

```
ARRAY TEXT(ColumnName;size)
```

Igualmente puede utilizar una lista:

```
LIST TO ARRAY("ListName";ColumnName)
```

***Advertencia**: cuando un list box contiene varias columnas de diferentes tamaños, sólo se muestra el número de elementos del array más pequeño (columna). Debe asegurarse de que cada array tenga el mismo número de elementos que los otros. Igualmente, si una columna de la list box está vacía (esto ocurre cuando el array asociado no fue declarado correctamente o dimensionado utilizando el lenguaje), la list box no muestra nada.*

List box de tipo selección

En este tipo de list box, cada columna puede estar asociada con un campo o una expresión. El contenido de cada fila se evalúa de acuerdo a una selección de registros: la selección actual de una tabla o una selección temporal.

Cuando la selección actual es la fuente de datos, todas las modificaciones realizadas en la base son reportadas automáticamente en el list box y viceversa. La selección actual es entonces siempre la misma en ambas ubicaciones. Note que los comandos

LISTBOX INSERT ROWS *y* ***LISTBOX DELETE ROWS*** *no pueden utilizarse con list boxes de tipo selección.*

Puede asociar una columna de list box a una expresión. La expresión puede estar basada en uno o más campos (por ejemplo [Empleados]Apellido+" "+[Empleados]Nombre) o simplemente en una fórmula (por ejemplo ***String(Milliseconds)****). La expresión también puede tener un método de proyecto, una variable o un elemento de array.*

El comando ***LISTBOX SET TABLE SOURCE*** *puede utilizarse para modificar la tabla asociada con el list box por programación.*

List boxes de tipo Colección o entity selection

En este tipo de list box, cada columna debe estar asociada a una expresión. El contenido de cada línea se evalúa para cada elemento de la colección o para cada entidad de la selección de entidades (entity selection).La colección debe haberse definido a través del comando ***C_COLLECTION****. Toda modificación realizada en la colección, por ejemplo, utilizando los diversos métodos del tema* ***Colecciones****, se transmite a través del list box y viceversa.*

Cada elemento de la colección o cada entidad está disponible como un objeto al que se puede acceder a través del comando ***This*** *.*

La expresión de la columna puede ser un método proyecto, una variable o cualquier fórmula, accediendo a cada objeto entidad o elemento de colección vía ***This****, por ejemplo* ***This.<propertyPath>*** *(o* ***This.value*** *en el caso de una colección de valores escalares). Puede utilizar los comandos* ***LISTBOX SET COLUMN FORMULA*** *y* ***LISTBOX INSERT COLUMN FORMULA*** *para modificar columnas por programación.*

Cuando la fuente de datos es una colección, cualquier modificación realizada en la colección, por ejemplo, utilizando los diversos métodos del tema [#title id="9128"/], se transmite a través del list box y viceversa.

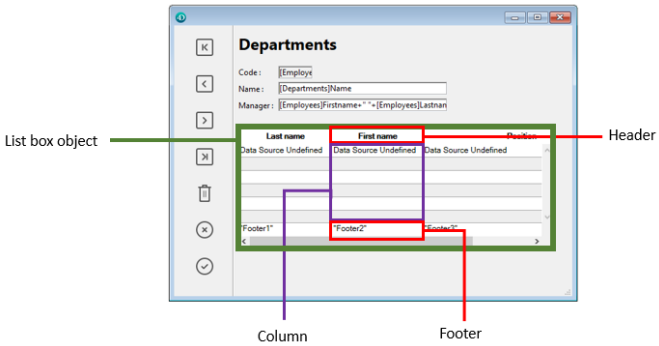
Cuando la fuente de datos es una selección de entidades, cualquier modificación realizada del lado de la lista se guarda automáticamente en la base de datos. Por otro lado, las modificaciones realizadas del lado de la base son visibles en el list box después de que las entidades tocadas hayan sido recargadas.

Objeto, columna, encabezado y pie

Un objeto List box está compuesto de cuatro tipos de elementos diferentes:

- *el* ***objeto*** *mismo,*
- *las* ***columnas****,*
- *los* ***encabezados*** *de las columnas (pueden mostrarse u ocultarse, se muestran por defecto),*
- *los* ***pies*** *de las columnas (pueden mostrarse u ocultarse, se ocultan por defecto),*

Estos elementos pueden seleccionarse individualmente en el editor de formularios. Cada uno tiene su propio nombre de objeto y nombre de variable y puede ser manejado por separado.



Por defecto, las columnas se llaman ***Columna1*** *a n y los encabezados* ***Título1*** *a n en el formulario, independientemente de los objetos list box. Note que por defecto, el mismo nombres se utiliza para los objetos y sus variables asociadas, a excepción de los pies (por defecto las variables están vacías para los pies, 4D utiliza las variables dinámicas).*
Cada tipo de elemento contiene características propias y características compartidas con los otros elementos. Por ejemplo, la fuente de los caracteres puede ser asignada globalmente al objeto list box o por separado a las columnas, a los encabezados y/o a los pies. Por otra parte, las propiedades de entrada sólo pueden definirse para las columnas.

*Estos principios se aplican a los comandos del tema "****Propiedades de los objetos****" que pueden utilizarse con los list box. Dependiendo de su funcionalidad, cada comando puede utilizarse con el list box, las columnas, los encabezados y/o los pies de las columnas. Para definir el tipo de elemento en el cual quiere trabajar, simplemente pase el nombre o la variable asociada a él.*

*La siguiente tabla detalla el alcance de cada comando del tema "****Propiedades de los objetos****" que puede utilizarse con los objetos de tipo list box:*

Comandos Prop. de los objetos	Objeto	Columna	Encabezado de columna	Pie de columna
OBJECT MOVE	X			
OBJECT GET COORDINATES	X	X	X	X
OBJECT SET RESIZING OPTIONS	X			
OBJECT GET RESIZING OPTIONS	X			
OBJECT GET BEST SIZE		X		
OBJECT SET FILTER		X		
OBJECT SET FORMAT		X		X
OBJECT SET ENTERABLE		X		
OBJECT SET LIST BY NAME		X		
OBJECT SET TITLE			X	
OBJECT SET COLOR	X	X	X	X
OBJECT SET RGB COLORS	X	X	X	X
OBJECT SET FONT	X	X	X	X
OBJECT SET FONT SIZE	X	X	X	X
OBJECT SET FONT STYLE	X	X	X	X
OBJECT SET HORIZONTAL ALIGNMENT	X	X	X	X
OBJECT Get horizontal alignment	X	X	X	X
OBJECT SET VERTICAL ALIGNMENT	X	X	X	X
OBJECT Get vertical alignment	X	X	X	X
OBJECT SET VISIBLE	X	X	X	X
OBJECT SET SCROLLBAR	X			
OBJECT SET SCROLL POSITION	X			
OBJECT SET HELP TIP	X		X	X
OBJECT Get help tip	X		X	X

***Nota**: con list boxes de tipo array, es posible especificar el estilo, color de fuente, color de fondo y visibilidad para cada línea por separado. Esto se administra por medio de arrays asociados con el list box en la lista de propiedades. Puede recuperar los nombres de estos arrays por programación utilizando el comando **LISTBOX GET ARRAYS**.*

List box y Lenguaje

Métodos de objeto

Es posible añadir un método de objeto al objeto list box en su conjunto y/o a cada columna de la list box.
Los métodos de objeto se llaman en la siguiente orden:

1. *Método de objeto de la columna*
2. *Método de objeto del list box*

El método de objeto de la columna recibe los eventos que ocurren en su encabezado y en su pie.

OBJECT SET VISIBLE y encabezados/pies

*Cuando el comando **OBJECT SET VISIBLE** se utiliza con un encabezado o un pie de list box, se utiliza en todos los encabezados o todos los pies del objeto List box, sin importar el elemento individual especificado por el comando.*
*Por ejemplo, la instrucción **OBJECT SET VISIBLE(*;"encabezado3";False)** ocultará todos los encabezados del objeto List box al cual pertenece el encabezado3 y no únicamente este encabezado.*
*Note que para poder manejar la visibilidad de estos objetos utilizando el comando **OBJECT SET VISIBLE**, debe haber sido mostrados en el list box a nivel del editor de formularios (la opción **Mostrar encabezados** y/o **Mostrar pies** debe estar seleccionada para el objeto).*

OBJECT Get pointer

*La función **OBJECT Get pointer** utilizada con la constante Object with focus u Object current (antiguas funciones **Focus object** y **Self**) puede ser utilizada en el método de objeto de un list box o de una columna de list box.*
Devuelven un puntero al list box, la columna(1) list box o la variable del encabezado en función del tipo de evento de formulario.
La siguiente tabla detalla este funcionamiento:

Evento	Objeto foco	Objeto actual
On Clicked	list box	columna
On Double Clicked	list box	columna
On Before Keystroke	columna	columna
On After Keystroke	columna	columna
On After Edit	columna	columna
On Getting Focus	columna o list box (*)	columna o list box (*)
On Losing Focus	columna o list box (*)	columna o list box (*)
On Drop	list box source	list box (*)
On Drag Over	list box source	list box (*)
On Begin Drag Over	list box	list box (*)
On Mouse Enter	list box (**)	list box (**)
On Mouse Move	list box (**)	list box (**)
On Mouse Leave	list box (**)	list box (**)
On Data Change	columna	columna
On Selection Change	list box (**)	list box (**)
On Before Data Entry	columna	columna
On Column Moved	list box	columna
On Row Moved	list box	list box
On Column Resize	list box	columna
On Open Detail	Nil	list box (**)
On Close Detail	Nil	list box (**)
On Header Click	list box	encabezado
On Footer Click	list box	footer
On After Sort	list box	encabezado

(*) Cuando el foco se modifica dentro de un list box, se devuelve a la columna un puntero. Cuando el foco se modifica a nivel del formulario, se devuelve un puntero al list box. En el contexto de un método de objeto de columna, se devuelve un puntero a la columna.

(**) No ejecutado en el contexto de un método de objeto de columna.

*(1) Cuando se devuelve un puntero a una columna, el objeto al cual se apunta depende del tipo del list box. Con un array de tipo list box, la función **OBJECT Get pointer** devuelve un puntero al array). El mecanismo de punteros de 4D permite conocer el número del elemento del array modificado. Por ejemplo, suponiendo que el usuario modificó la quinta fila de la columna col2:*

```
$Column:=OBJECT Get pointer(Object with focus)
  ` $Column contiene un puntero a col2
$Fila:=$Column->  `$Fila igual a 5
```

*En el caso de un list box de tipo selección, la función **OBJECT Get pointer** devuelve:*

- *Para una columna asociada a un campo, un puntero al campo asociado,*
- *Para una columna asociada a una variable, un puntero a la variable,*
- *Para una columna asociada a una expresión, el puntero **Nil**.*

OBJECT SET SCROLL POSITION

*El comando **OBJECT SET SCROLL POSITION** (tema "Propiedades de los objetos") puede utilizarse con un objeto de tipo list box. Ese comando permite desplazar las líneas de la list box para mostrar la primera línea seleccionada o una línea específica.*

EDIT ITEM

*El comando **EDIT ITEM** (tema "Gestión de entrada") le permite pasar a modo edición una celda de un objeto list box.*

REDRAW

*Cuando se aplica a un listbox en modo selección, el comando **REDRAW** (tema "**Interfaz de usuario**") dispara la actualización de los datos mostrados en el list box.*

***Nota:** el comando **REDRAW** no es soportado con los list box de tipo entity selection.*

Displayed line number

*El comando **Displayed line number** (tema "**Selecciones**") funciona en el contexto del evento de formulario On Display Detail para un objeto list box.*

Form events

*Los eventos de formulario específicos están destinados a facilitar la gestión programada del list box, en particular con lo relacionado a las operaciones de arrastrar y soltar y ordenar. Para mayor información, consulte la descripción del comando **Form event**.*

Drag and drop

*La gestión de arrastrar y soltar de datos en los list boxes está soportada por los comandos **Drop position** y **DRAG AND DROP PROPERTIES**. Estos comandos han sido adaptados especialmente para los list boxes.*

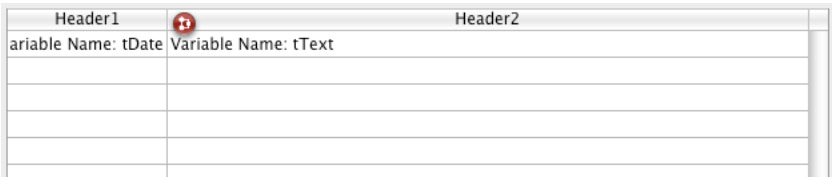
*Tenga cuidado en no confundir arrastrar y soltar con mover filas y columnas, soportado por los comandos **LISTBOX MOVED ROW NUMBER** y **LISTBOX MOVED COLUMN NUMBER**.*

Gestión de entrada

Para que una celda de list box sea editable, ambas de las siguientes condiciones deben cumplirse:

- *La columna de la celda debe haberse definido como **Editable** (de lo contrario, las celdas de la columna no serán editables).*
- *En el evento formulario On Before Data Entry, $0 no devuelve -1.*
  *Cuando el cursor llega a la celda, el evento On Before Data Entry se genera en el método de la columna. Si, en el contexto de este evento, $0 toma el valor -1, la celda se considera como no editable. Si el evento se generó luego de presionar **Tab** o **Mayús+Tab**, el foco va a la celda siguiente o anterior respectivamente. Si $0 no vale -1 (por defecto $0 es 0), la celda es editable y pasa a modo edición.*

Imaginemos por ejemplo un list box que contiene dos arrays, de tipo fecha y texto. El array fecha no es editable pero el array texto es editable si la fecha no ha pasado.



Este es el método de la columna arrText:

```
Case of
   :(Form event=On Before Data Entry) // una celda obtiene el foco
      LISTBOX GET CELL POSITION(*;"lb";$col;$row)
 // identificación de la celda
      If(arrDate{$row}<Current date) // si la fecha es anterior a la de hoy
         $0:=-1 // la celda NO es editable
      Else
 // de lo contrario, la celda es editable
      End if
End case
```

***Nota:** a partir de 4D v13, el evento On Before Data Entry se devuelve antes de On Getting Focus.*
Con el fin de preservar la coherencia de los datos para los list box de tipo selección, todo registro/entidad modificado(a) se guarda tan pronto como se valida la celda, es decir:

- *cuando la celda está desactivada (el usuario presiona la pestaña, clics, etc.)*
- *cuando el listbox ya no tiene el foco,*
- *cuando el formulario ya no tiene el primer plano.*

La secuencia típica de eventos generada durante la entrada o la modificación de datos es la siguiente:

Acción	***Tipos de listbox***	***Secuencia de eventos***
Una celda pasa a modo edición	*Todos*	*On Before Data Entry*
	Todos	*On Getting Focus*
Su valor se modifica	*Todos*	*On Before Keystroke*
	Todos	*On After Keystroke*
	Todos	*On After Edit*
Un usuario valida y sale de la celda	*List box selección*	***Save***
	List box selección de registros	*Dispara On saving an existing record (si definido)*
	List box selección	*On Data Change(*)*
	List box Entity selection	*La entidad se guarda con la opción automerge, bloqueo optimista (ver [#title id="9310"/]). En caso de guardar correctamente, la entidad se actualiza con la última actualización realizada. Si la operación de guardar falla, se muestra un error*
	Todos	*On Losing Focus*

(*) Con los list box de tipo entity selection, en el evento On Data Change:

- *el objeto **Elemento actual** (ver **Tema Fuente de datos**) contiene el valor antes de modificación.*
- *el objeto **This** contiene el valor modificado.*

***Nota:** la entrada de datos en los list boxes de tipo colección/entity selection tiene una limitación cuando la expresión se evalúa como **null**. En este caso, no es posible editar o eliminar el valor **null** en la celda.*

Gestión de ordenaciones

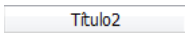
Por defecto, la list box administra automáticamente la ordenación estándar de columnas en caso de clic en el encabezado. Una ordenación estándar es una ordenación alfanumérica de valores de la columna, de forma alterna ascendente/descendente con cada clic. Todas las columnas siempre se sincronizan automáticamente.
Puede evitar las ordenaciones de usuario estándar deseleccionando la propiedad "Ordenable" del list box.

Por defecto, el list box maneja automáticamente las ordenaciones estándar de columnas en caso de clic en el encabezado. Una ordenación estándar es una ordenación alfanumérica de los valores de la columna, alternamente ascendente/descendente con cada clic sucesivo. Todas las columnas siempre se sincronizan automáticamente.
Usted puede prohibir ordenar al usuario estándar deseleccionando la propiedad "Ordenable" para el listbox.
El desarrollador puede configurar ordenaciones personalizadas utilizando el comando ***LISTBOX SORT COLUMNS*** *y/o combinando los eventos de formulario On Header Click y On After Sort (ver el comando* ***Form event****) y los comandos 4D de gestión de arrays.*

Nota*: la propiedad "Ordenable" únicamente afecta la ordenación del usuario estándar; el comando* ***LISTBOX SORT COLUMNS*** *no tiene en cuenta esta propiedad.*

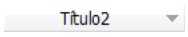
El valor de la variable asociada al título de la columna le permite administrar información adicional: la ordenación actual de la columna (lectura) y la visualización de la flecha de ordenación.

- *Si la variable vale 0, la columna no se ordena y no se muestra la flecha de ordenación;*



- *Si la variable vale 1, la columna se ordena de forma ascendente y se muestra la flecha de ordenación;*

- *Si la variable vale 2, la columna se ordena de forma descendente y se muestra la flecha de ordenación.*



Es posible fija el valor de la variable (por ejemplo, ***Titulo2:=2****) para "forzar" la visualización de la flecha de ordenación. La ordenación de la columna misma no se modifica en este caso; está en manos del desarrollador manejarlo.*

Nota*: el comando* ***OBJECT SET FORMAT*** *ofrece un soporte específico para los íconos en los encabezados de list box, útil cuando desea trabajar con un icono de ordenación personalizado.*

Gestión de las selecciones

La gestión de selecciones se efectúa de manera diferente para los list box de tipo array, selección de registros, o colección/entity selection.

- ***List box de tipo selección:*** *las selecciones son administradas por intermedio de un conjunto llamado por defecto $ListboxSetN (N comienza en 0 y se incrementa en función del número de list box en el formulario), puede modificarlo si es necesario. Este conjunto se define en las propiedades del list box. Es mantenido automáticamente por 4D: si el usuario selecciona una o más filas en el list box, el conjunto se actualiza inmediatamente. Por otra parte, también es posible utilizar los comandos del tema "Conjuntos" con el fin de modificar por programación la selección del list box.*
- ***List box de tipo colección/Entity selection****: las selecciones son manejadas por medio de las propiedades de list box dedicadas: Elemento actual es un objeto que recibirá el elemento/entidad seleccionado, Elementos seleccionados es una colección de elementos seleccionados, y Posición elemento actual devuelve la posición del elemento o entidad seleccionado. Para mayor información, consulte la sección* ***Tema Fuente de datos****.*
- ***List box de tipo array:*** *el comando* ***LISTBOX SELECT ROW*** *puede utilizarse para seleccionar por programación una o más líneas de list box.*

La variable asociada al objeto List box se utiliza para obtener, fijar o almacenar las selecciones de filas del objeto.

Esta variable corresponde a un array de booleanos que es creada y mantenida automáticamente por 4D. El tamaño de este array se determina por el tamaño del list box: contiene el mismo número de elementos que el array más pequeño asociado a las columnas.

Cada elemento de este array contiene ***True*** *si la fila correspondiente se selecciona y de lo contrario* ***False****. 4D actualiza los contenidos de este array dependiendo de las acciones del usuario. A la inversa, usted puede modificar el valor de los elementos de este array para modificar la selección en el list box.*

Por otra parte, no es posible insertar o borrar filas en este array; tampoco es posible digitar filas nuevamente.

Nota*: el comando* ***Count in array*** *es útil para conocer el número de líneas seleccionadas.*

Por ejemplo, este método permite invertir la selección de la primera línea del list box (tipo array):

```
ARRAY BOOLEAN(tBListBox;10)
\\tBListBox es el nombre de la variable asociada al list box en el formulario
If(tBListBox{1}=True)
   tBListBox{1}:=False
Else
   tBListBox{1}:=True
End if
```

Si seleccionó la opción ***Ocultar la selección resaltada*** *para un list box, necesitará volver las selecciones de list box visibles utilizando las opciones de interfaz disponibles. Para más información sobre cómo hacer esto, consulte* ***Personalizar la apariencia de las selecciones****.*

Nota*: las especificaciones de la gestión de las selecciones en los list box en modo jerárquico se detallan en la sección [#title id="2774"/].*

Impresión de list boxes

Es posible imprimir list boxes a partir de 4D v12. Están disponibles dos modos de impresión: modo previsualización, el cual puede permite imprimir un list box como un objeto de formulario y el modo avanzado, permite controlar la impresión del objeto

list box mismo dentro del formulario. Note que la apariencia "Impresión" está disponible para los objetos list box en el editor de formularios.

Modo previsualización

*La impresión de un list box en modo previsualización consiste en imprimir directamente el list box con el formulario que lo contiene utilizando los comandos de impresión estándar o el comando de menú **Print**. El list box se imprime como está en el formulario. Este modo no permite controlar con precisión la impresión del objeto; en particular, no permite imprimir todas las líneas de un list box que contenga más líneas de las que pueda mostrar.*

Modo avanzado

*En este modo, la impresión de los list box se efectúa por programación, vía el comando **Print object** (los formularios proyecto y los formularios tabla se tienen en cuenta). El comando **LISTBOX GET PRINT INFORMATION** se utiliza para controlar la impresión del objeto.*

En este modo:

- *La altura del objeto list box se reduce automáticamente cuando el número de líneas a imprimir es menor a la altura original del objeto (no hay líneas "vacías" impresas). Por el contrario, la altura no aumenta automáticamente en función del contenido del objeto. El tamaño del objeto efectivamente impreso puede obtenerse vía el comando **LISTBOX GET PRINT INFORMATION**.*
- *El objeto list box se imprime "tal cual", en otras palabras, teniendo en cuenta sus parámetros de visualización actuales: visibilidad de encabezados y rejillas, líneas mostradas y ocultas, etc.*
  *Estos parámetros también incluyen la primera línea a imprimir: si llama al comando **OBJECT SET SCROLL POSITION** antes del lanzar la impresión, la primera línea impresa en el list box será la designada por el comando.*
- *Un mecanismo automático facilita la impresión de los list box que contienen más líneas de las que es posible mostrar: llamadas sucesivas a **Print object** permiten imprimir un nuevo conjunto de líneas cada vez. El comando **LISTBOX GET PRINT INFORMATION** permite controlar el estado de la impresión.*

Gestión de estilos y de colores

Hay varios modos distintos de configurar los colores de fondo, colores de fuente y estilos de fuente para los list boxes:

- *a nivel de las propiedades del objeto list box,*
- *a nivel de las propiedades de la columna,*
- *utilizando arrays o métodos para el list box y/o para cada columna,*
- *a nivel del texto de cada celda (si texto multiestilo).*

Se observan principios de prioridad y de herencia.

propiedades del objeto list box o columna, el uso de arrays o métodos para el list box o por columna y definición a nivel de celdas (si son de texto multiestilo).

Prioridad

Cuando una misma propiedad se define en más de un nivel, el orden de prioridad es el siguiente:

Prioridad elevada	*Celda (si texto multiestilo)*
	Array/métodos de columna
	Array/métodos de listbox
	Propiedades de la columna
Prioridad baja	*Propiedades del list box*
low priority	*Expresión Meta Info (list box de tipo colección o entity selection)*

Por ejemplo, si define un estilo de caracteres en las propiedades del list box y otro utilizando un array de estilo para la columna, se tendrá en cuenta este último.

Dado un list box cuyas filas tienen un color alterno gris/gris claro, definido en las propiedades del list box. Un array de color de fondo también se ha definido para el list box con el fin de cambiar el color de las filas en las que al menos un valor es negativo por naranja:

```
<>_BgndColors{$i}:=0x00FFD0B0 // orange
<>_BgndColors{$i}:=-255 // valor por defecto
```

Header1	Header2	Header3
21483	9031	27290
24151	21990	-923
21351	2982	18009
8089	12898	20941
13001	-802	22059
4321	16826	11303
24082	26214	22380
16680	23651	20403
-2678	24818	29896
25639	2691	9687
28794	26941	21486
26083	21092	13476
27928	4092	15441
19987	28211	21191
7996	6300	4089
-1063	13388	23683
13008	7470	19897
5388	26918	13547
28559	27007	8365
28454	22646	13824

A continuación, vamos a colorear los valores negativos de color naranja oscuro. Para ello , se define un array de color de fondo para cada columna, por ejemplo <>_BgndColor_1, <>_BgndColor_2 and <>_BgndColor_3. Los valores de estos arrays serán prioritarios sobre los definidos en las propiedades del list box, así como las del array general de color de fondo:

```
<>_BgndColorsCol_3{2}:=0x00FF8000 // naranja oscuro
<>_BgndColorsCol_2{5}:=0x00FF8000
<>_BgndColorsCol_1{9}:=0x00FF8000
<>_BgndColorsCol_1{16}:=0x00FF8000
```

Header1	Header2	Header3
21483	9031	27290
24151	21990	-923
21351	2982	18009
8089	12898	20941
13001	-802	22059
4321	16826	11303
24082	26214	22380
16680	23651	20403
-2678	24818	29896
25639	2691	9687
28794	26941	21486
26083	21092	13476
27928	4092	15441
19987	28211	21191
7996	6300	4089
-1063	13388	23683
13008	7470	19897
5388	26918	13547
28559	27007	8365
28454	22646	13824

*Puede obtener el mismo resultado utilizando los nuevos comandos **LISTBOX SET ROW FONT STYLE** y **LISTBOX SET ROW COLOR**. Tienen la ventaja de que le permite evitar tener que predefinir los arrays de estilo/color de las columnas: son creados dinámicamente por los comandos.*

Herencia

Para cada atributo (estilo, color y color de fondo), la herencia se aplica cuando se utiliza el valor por defecto:

- *para los atributos de las celdas: valores de atributos de líneas*
- *para los atributos de las líneas: valores de los atributos de las columnas*
- *para los atributos de las columnas: valores de los atributos del listbox*

De esta manera, si quiere que un objeto herede el valor del atributo de un nivel superior, pase la constante lk inherited (valor por defecto) al comando de definición o directamente en el elemento del array de estilo/color correspondiente.

Dado un listbox que contiene una estilo de fuente estándar con colores alternativos:

standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard

Puede realizar las modificaciones siguientes:

- *cambiar el fondo de la fila 2 a rojo con la propiedad **Array colores de fondo** del objeto list box,*
- *cambiar el estilo de la fila 4 a itálica por la propiedad **Array de estilos** del objeto list box,*

- *dos elementos de la columna 5 se pasan a negrita por la propiedad **Array de estilos** del objeto columna 5,*
- *los elementos de la fila 2 de las columnas 1 y 2 se cambian a fondo azul utilizando la propiedad Array colores de fondo de los objetos columna 1 y 2:*

standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	**standard**	standard
standard	*standard*	*standard*	*standard*	**standard**	*standard*
standard	standard	standard	standard	standard	standard
standard	standard	standard	standard	standard	standard

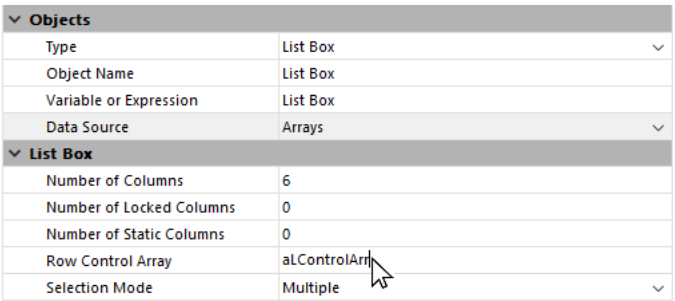
Puede realizar las modificaciones siguientes:

- *pase la constante lk inherited en los elementos 2 de los arrays de fondo de las columnas 1 y 2: a continuación, heredan el color de fondo rojo de la línea.*
- *pase la constante lk inherited en los elementos 3 y 4 de los arrays de estilo de la columna 5: entonces heredan el estilo estándar, excepto el elemento 4, que cambia a itálica como se define en el array de estilo del list box*
- *pase la constante lk inherited en el elemento 4 del array de estilo del list box con el fin de suprimir el estilo itálica.*
- *pase la constante lk inherited en el elemento 2 del array de colores de fondo del list box con el fin de restaurar el color alterno original del list box.*

Gestión de visualización de las líneas

Puede definir las propiedades de interfaz "oculta", "desactivada" y "seleccionable" para cada línea de list box de tipo array.

*Estas funcionalidades se pueden manejar por medio del **Array de control de líneas**, el cual puede designar utilizando el comando **LISTBOX SET ARRAY** o vía la Lista de propiedades:*



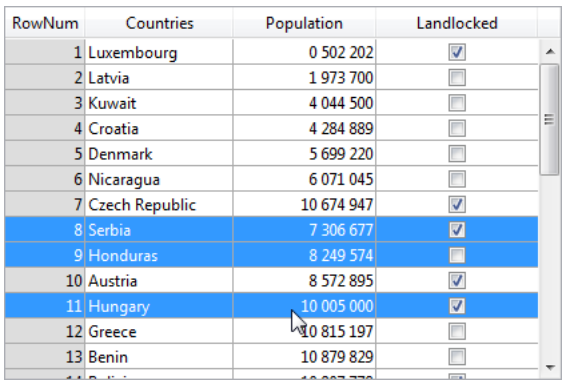
*Un array de control de líneas debe ser de tipo Entero largo e incluir el número de líneas que el list box. Para obtener más información, consulte la sección **Propiedades específicas de los list box**.*

*Cada elemento del **Array de control de líneas** define el estado de la interfaz de la línea correspondiente en el list box. Tres propiedades de interfaz están disponibles usando las constantes del tema "List box":*

Constante	***Tipo***	***Valor***	***Comentario***
lk row is disabled	*Entero largo*	*2*	*La línea correspondiente se desactiva. El texto y los controles tales como casillas de selección se atenúan o se ponen grises. Las áreas de entrada de texto editables ya no son editables. Valor por defecto: Activado*
lk row is hidden	*Entero largo*	*1*	*La línea correspondiente está oculta. Ocultar las líneas sólo afecta la visualización del list box. Las líneas ocultas siguen presentes en los arrays y pueden ser manipuladas por programación. Los comandos del lenguaje, más concretamente **LISTBOX Get number of rows** o [#cmd id="971"/], no tienen en cuenta el estado visible/oculto de las líneas. Por ejemplo, en un list box con 10 líneas donde las primeras 9 líneas se ocultan, **LISTBOX Get number of rows** devuelve10. Desde el punto de vista del usuario, la presencia de líneas ocultas en un list box no es visiblemente discernible. Sólo líneas visibles pueden ser seleccionadas (por ejemplo utilizando el comando Seleccionar todo). Valor por defecto: Visible*
lk row is not selectable	*Entero largo*	*4*	*La línea correspondiente no se puede seleccionar (resaltarla no es posible). Las áreas de entrada de texto editables ya no son editables a menos de que la opción "Editar en clic único" esté activada. Los controles tales como casillas de selección y listas siguen siendo funcionales sin embargo. Este parámetro se ignora si el modo de selección del list box es "Ninguno". Valor por defecto: seleccionable*

Para cambiar el estado de una línea, sólo tiene que pasar las constantes apropiadas en el elemento correspondiente. Por ejemplo, si o quiere que la línea #10 sea seleccionable, puede escribir:

```
aLControlArr{10}:=lk row is not selectable
```



Puede definir varias propiedades de la interfaz a la vez:

```
aLControlArr{8}:=lk row is not selectable+lk row is disabled
```

RowNum	Countries	Population	Landlocked
1	Luxembourg	0 502 202	☑
2	Latvia	1 973 700	☐
3	Kuwait	4 044 500	☐
4	Croatia	4 284 889	☐
5	Denmark	5 699 220	☐
6	Nicaragua	6 071 045	☐
7	Czech Republic	10 674 947	☑
8	Serbia	7 306 677	☑
9	Honduras	8 249 574	☐
10	Austria	8 572 895	☑
11	Hungary	10 005 000	☑
12	Greece	10 815 197	☐
13	Benin	10 879 829	☐

Tenga en cuenta que la definición de una propiedad para un elemento anula cualquier otro valor de este elemento (si no reinicializa). Por ejemplo:

```
aLControlArr{6}:=lk row is disabled+lk row is not selectable  //define la línea 6 como desactivada Y no seleccionable
aLControlArr{6}:=lk row is disabled  //define la línea 6 como desactivado pero seleccionable nuevamente
```

Personalizar la apariencia de las selecciones

*Cuando se selecciona la opción **Ocultar el resaltado de selección**, debe hacer que las selecciones del list box estén visibles utilizando las opciones de interfaz disponibles. Dado que las selecciones todavía están totalmente gestionadas por 4D, esto significa:*

- *Para los list boxes de tipo array, debe analizar la variable de array booleana asociada con el list box para determinar qué filas están seleccionadas o no.*
- *Para los list boxes de tipo selección, debe comprobar si el registro actual (fila) pertenece al conjunto especificado en la propiedad **Resaltar conjunto** del list box.*

A continuación, puede definir colores de fondo específicos, colores de fuente y/o estilos de fuente por programación para personalizar la apariencia de las filas seleccionadas. Esto se puede hacer utilizando matrices o expresiones, dependiendo del tipo de list box que se muestre (consulte las siguientes secciones).

***Nota:** puede utilizar la constante lk inherited para imitar la apariencia actual del list box (e.g., font color, background color, font style, etc.). (por ejemplo, color de fuente, color de fondo, estilo de fuente, etc.). Para más información acerca de cómo funciona la constante lk inherited, consulte la sección **Herencia** "Gestión de objetos de list box" en el Manual de Diseño.*

List boxes de tipo Selección

*Para determinar qué filas están seleccionadas, debe comprobar si están incluidas en el conjunto indicado en la propiedad **Resaltar conjunto** del list box:*

A continuación, puede definir la apariencia de las filas seleccionadas utilizando una o más de las siguientes expresiones especificadas en la Lista de propiedades:

- ***Expresión de color de fondo** (tema Fondo y Borde)*
- ***Expresión de color de fuente** (tema Texto)*
- ***Expresión de estilo** (tema Texto)*

***Nota:** tenga en cuenta que las expresiones se revalúan automáticamente cada vez que:*

- *La selección del list box cambia.*
- *El list box obtiene o pierde el foco.*
- *La ventana del formulario que contiene el list box se convierte en, o deja de ser, la ventana frontal.*

Limitación con list boxes jerárquicos

*Las líneas de ruptura no se pueden resaltar cuando la opción **Ocultar el resaltado de selección** está seleccionada.*

Dado que no es posible tener colores distintos para los encabezados del mismo nivel, no hay forma de resaltar una línea de ruptura específica por programación.

List boxes de tipo Array

Debe analizar la variable de array booleana asociada con el list box para determinar si las líneas están seleccionadas o no:

A continuación, puede definir la apariencia de las líneas seleccionadas utilizando una o más de las siguientes arrays especificadas en la Lista de propiedades:

- ***Array de color de fondo de línea** (Tema Fondo y Borde)*

- ***Array de color de fuente de línea*** *(tema Texto)*
- ***Array de estilo de línea*** *(tema Texto)*

Tenga en cuenta que los arrays de list box utilizados para definir la apariencia de líneas seleccionadas deben ser recalculados durante el evento de formulario On Selection Change; sin embargo, también puede modificar estos arrays basándose en los siguientes eventos de formulario adicionales:

- *On Getting Focus (propiedad list box)*
- *On Losing Focus (propiedad list box)*
- *On Activate (propiedad formulario)*
- *On Deactivate (propiedad formulario)*

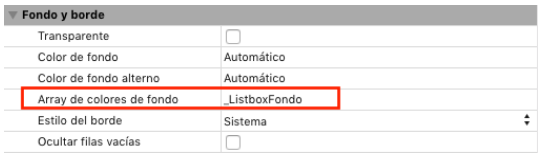
... dependiendo de cómo desea representar visualmente los cambios de enfoque en las selecciones.

Ejemplo

Usted ha optado por ocultar el resaltado del sistema y desea mostrar selecciones de list box con un color de fondo verde, como se muestra aquí:

Category	ID	Reference	Value
Alpha	215	5A0DF64-EC5-955F7EA-BD284E4-8A	15,425
Bravo	196	D9E3484-547-AEECB8B-B1808FF-A6	4,592
Alpha	205	3295824-3AB-B48B870-2074C57-B9	-3,672
Charlie	197	B3800C4-C64-A6C95CB-ED27729-B5	16,212
Echo	214	835F344-8EE-B422E66-5C52074-01	-12,332
Alpha	200	46784B4-B20-AE2E51D-0159EA4-D0	1,283
Delta	213	11F5FD4-E48-9BD6E93-E8B9F82-59	13,236
Delta	203	3E80494-879-9F2CEC2-4008AA4-F5	-12,231
Charlie	202	015D694-9C8-91113AA-B8A51A1-52	27,100
Bravo	211	E998AC4-FAE-93BE025-E4CA634-E8	2,630
Charlie	207	B19F244-A30-A03B668-C407B43-D4	16,677
Delta	208	41B1DE4-D29-BC8E7BF-5062D92-B7	-14,759
Echo	199	7005654-722-926CDCE-D8E1BBD-83	23,952
Delta	198	0AD0734-0C7-BA8168E-A0AB67A-1A	-19,758
Alpha	210	6F46794-0D6-AF0E61A-D43231E-3E	24,342
Bravo	201	00A8334-B4B-8419285-8772CFB-B4	-3,657
Charlie	212	9EF2FD4-B1B-97138DE-B3750BA-FB	-4,850
Echo	209	FD05424-365-B8DB0C2-91098A8-80	2,941
Echo	204	7473724-2FA-82F49A5-010BDED-98	22,200
Bravo	206	3537D34-A9A-AE41C4E-34EC5B6-43	1,205

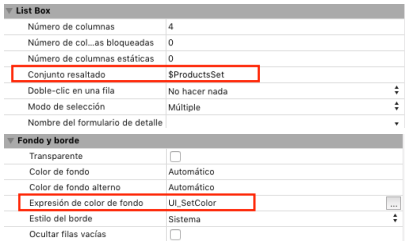
Para un list box de tipo array, debe agregar un método de objeto para actualizar la propiedad ***Array de fondo de línea****:*



En este método de objeto, puede escribir:

```
Case of
   :(Form event=On Selection Change)
      $n:=Size of array(LB_Arrays)
      ARRAY LONGINT(_ListboxBackground;$n) // <span id="result_box" lang="es"><span title="ARRAY
LONGINT(_ListboxBackground;$n) // row background colors ">colores de fondo de línea</span></span>
      For($i;1;$n)
         If(LB_Arrays{$i}=True) // seleccionado
            _ListboxBackground{$i}:=0x0080C080 // <span id="result_box" lang="es"><span
title="_ListboxBackground{$i}:=0x0080C080 // green background ">fondo verde</span></span>
         Else // no seleccionado
            _ListboxBackground{$i}:=lk inherited
         End if
      End for
End case
```

Para un list box de tipo de selección, para producir el mismo efecto, puede utilizar un método para actualizar la ***Expresión de color de fondo*** *basada en el conjunto especificado en la propiedad* ***Resaltar conjunto****. Por ejemplo:*



En el método ***UI_SetColor****, puede escribir:*

```
If(Is in set("$SampleSet"))
   $color:=0x0080C080 // fondo verde
Else
   $color:=lk inherited
End if

$0:=$color
```

Visualización del resultado de una petición SQL en un list box

*Es posible poner directamente el resultado de una petición SQL en un list box de tipo array. Esta función ofrece un medio rápido de visualizar el resultado de peticiones SQL. Sólo pueden utilizarse las peticiones de tipo **SELECT**. Este mecanismo no puede utilizarse con una base SQL externa.*

Esto funciona de acuerdo a los siguientes principios:

- *Cree el list box que recibirá los resultados de la petición. La fuente de datos del list box debe ser **Arrays**.*
- *Ejecute la petición SQL de tipo **SELECT** y asigne el resultado a la variable asociada al list box. Puede utilizar las palabras claves **Begin SQL**/**End SQL** (ver el manual de Lenguaje de 4D).*
- *Las columnas del list box son ordenables y modificables por el usuario.*
- *Cada nueva ejecución de una petición **SELECT** con la list box provoca la reinicialización de las columnas (no es posible llenar el mismo list box progresivamente utilizando varias peticiones **SELECT**).*
- *Se recomienda dar al list box el mismo número de columnas que las que tendrá en el resultado de petición SQL. Si el número de columnas del list box es inferior al del necesario para la petición **SELECT**, las columnas se añaden automáticamente. Si el número de columnas del list box es superior al necesario para la petición **SELECT**, se ocultan las columnas innecesarias.*
  ***Nota:** las columnas añadidas automáticamente están relacionadas con las **Variables dinámicas** de tipo array. Estos arrays dinámicos permanecen siempre y cuando el formulario exista. Una variable dinámica se crea igualmente para cada encabezado. Cuando se llama el comando **LISTBOX GET ARRAYS**, el parámetro arrVarCols contiene los punteros a los arrays dinámicos y el parámetro arrVarEncabezados contiene los punteros a las variables de encabezados dinámicos. Si una columna añadida es por ejemplo la quinta columna, su nombre es sql_column5 y su nombre de encabezado es sql_header5.*
- *En modo interpretado, los arrays existentes utilizados por el list box pueden redigitarse automáticamente de acuerdo a los datos enviados por la petición SQL*

Ejemplo
Queremos recuperar todos los campos de la tabla PERSONAS y ubicar su contenido en el list box cuyo nombre de variable es vlistbox. En el método de objeto de un botón (por ejemplo), es suficiente escribir:

```
Begin SQL
   SELECT * FROM PEOPLE INTO <<vlistbox>>
End SQL
```

Gestión de list box jerárquicos

4D le permite especificar y usar list box jerárquicos. Un list box jerárquico es un list box en el cual el contenido de la primera columna aparece en forma jerárquica. Este tipo de representación se adapta a la presentación de la información, incluyendo los valores repetidos y/o jerárquicamente dependientes (país/región/ciudad...).

*Sólo los **list box de tipo array** pueden ser jerárquicos.*

*Los list boxes de tipo jerárquico son una forma de representación particular de los datos pero no modifican la estructura de estos datos (los arrays). Los list box jerárquicos se llenan y manejan exactamente del mismo modo que los list box regulares (ver **Gestión programada de los objetos de tipo List box**).*

Para especificar un list box jerárquico, hay tres posibilidades diferentes:

- *configurar manualmente los elementos jerárquicos utilizando la lista de propiedades del editor de formularios o utilizando el menú emergente de gestión de list box. Estos puntos se tratan en el Manual de Diseño de 4D.*
- *utilizar los comandos **LISTBOX SET HIERARCHY** y **LISTBOX GET HIERARCHY**.*

Cuando se abre por primera vez un formulario que contiene un list box jerárquico, por defecto todas las líneas se despliegan. Una línea de ruptura y un "nodo" jerárquico se agregan automáticamente al list box cuando los valores se repiten en los arrays. Por ejemplo, imagine un list box que contiene cuatro arrays especificando las ciudades, cada ciudad caracterizada por un país, una región, un nombre y un número de habitantes:

País	Región	Ciudad	Población
Francia	Brittany	Rennes	200000
Francia	Brittany	Quimper	80000
Francia	Brittany	Brest	120000
Francia	Normandy	Caen	75000
Francia	Normandy	Deauville	35000

Si este list box se muestra en forma jerárquica (los tres primeros arrays se incluyen en la jerarquía), se obtiene:

País	Población
⊟ Francia	
⊟ Brittany	
Rennes	200000
Quimper	80000
Brest	120000
⊟ Normandy	
Caen	75000
Deauville	35000

Los arrays no se ordenan antes de la construcción de la jerarquía. Si, por ejemplo, un array contiene los datos AAABBAACC, la jerarquía obtenida será:

> *A*
> *B*
> *A*
> *C*

*Para expandir o contraer un "nodo" jerárquico, simplemente haga clic en él. Si presiona **Alt+clic** (Windows) u **Opción+clic** (Mac OS) en el nodo, todos sus sub-elementos se expandirán o contraerán. Estas operaciones también pueden realizarse por programación utilizando los comandos **LISTBOX EXPAND** y **LISTBOX COLLAPSE**.*

Gestión de selecciones y de posiciones

Un list box jerárquico muestra un número variable de líneas en la pantalla de acuerdo al estado desplegado/contraído de los nodos jerárquicos. Esto no significa, sin embargo que el número de líneas de los arrays varíe. Sólo se modifica la visualización, no los datos.

Es importante entender este principio porque la gestión programada de list box jerárquicos se basa siempre en los datos de los arrays, no en los datos mostrados. En particular, las líneas de ruptura añadidas automáticamente no se tienen en cuenta en los arrays de opciones de visualización (consulte la sección "Gestión de líneas de ruptura").

Echemos un vistazo por ejemplo a los siguientes arrays:

France	Brittany	Brest
France	Brittany	Quimper
France	Brittany	Rennes

Si estos arrays están representados jerárquicamente, la línea "Quimper" no se mostrará en la segunda línea, sino en la cuarta, debido a las dos líneas de ruptura añadidas:

France
Brittany
Brest
Quimper
Rennes

Independientemente de cómo los datos se muestran en el list box (jerárquicamente o no), si quiere cambiar la línea que contiene "Quimper" a negrita, debe utilizar la instrucción Style{2} = bold. Sólo se tiene en cuenta la posición de la línea en los arrays.

Este principio se aplica para los arrays internos que se pueden utilizar para administrar:

- *colores*
- *colores de fondo*
- *estilos*
- *líneas ocultas*
- *selecciones*

Por ejemplo, si desea seleccionar la fila que contiene Rennes, debe pasar:

```
->MiListbox{3}:=True
```

Representación no-jerárquica:

France	Brittany	Brest
France	Brittany	Quimper
France	Brittany	Rennes

Representación jerárquica:

France
Brittany
Brest
Quimper
Rennes

***Nota:** si una o más líneas se ocultan porque sus padres están contraídos, no se seleccionan más. Sólo las líneas que son visibles (ya sea directamente o por desplazamiento) pueden seleccionarse. En otras palabras, las líneas no pueden estar a la vez ocultas y seleccionadas.*

*Al igual que con las selecciones, el comando **LISTBOX GET CELL POSITION** devuelve los mismos valores para un list box jerárquico y un list box no jerárquico. Esto significa que en los dos ejemplos a continuación, **LISTBOX GET CELL POSITION** devolverá la misma posición: (3;2).*

Representación no-jerárquica:

France	Brittany	Brest	120000
France	Brittany	Quimper	80000
France	Brittany	Rennes	200000
France	Normandy	Caen	75000

Representación jerárquica:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000

Gestión de líneas de ruptura

*Si el usuario selecciona una línea de ruptura, **LISTBOX GET CELL POSITION** devuelve la primera ocurrencia de la línea en el array correspondiente. En el siguiente caso:*

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000

*... **LISTBOX GET CELL POSITION** devuelve (2;4). Para seleccionar una línea de ruptura por programación, debe utilizar el comando **LISTBOX SELECT BREAK**.*

Las líneas de ruptura no se tienen en cuenta en los arrays internos utilizados para administrar la apariencia gráfica de los list box (estilos y colores). Sin embargo es posible modificar estas características para las líneas de ruptura vía los comandos de gestión gráfica para objetos (tema). Sólo debe ejecutar los comandos apropiados en los arrays que constituyen la jerarquía.

Dado por ejemplo el siguiente list box (los nombres de los arrays asociados se especifican entre paréntesis):

Representación no-jerárquica:

(T1)	(T2)	(T3)	(T4)	(tStyle)	(tColor)
France	Brittany	Brest	120000	*Normal*	*0*
France	Brittany	Quimper	80000	*Underline*	*0*
France	Brittany	Rennes	200000	*Normal*	*0xFF0000*
France	Normandy	Caen	220000	*Normal*	*0*
France	Normandy	Deauville	4000	*Normal*	*0*

Representación jerárquica:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000
Deauville	4000

En modo jerárquico, los niveles de ruptura no son tenidos en cuenta por los arrayas de modificación de estilo llamados tStyle y tColors. Para modificar el color o el estilo de los niveles de ruptura, debe ejecutar las siguientes instrucciones:

```
OBJECT SET RGB COLORS(T1;0x0000FF;0xB0B0B0)
OBJECT SET FONT STYLE(T2;Bold)
```

Nota: *en este contexto, sólo la sintaxis que utiliza la variable array puede funcionar con los comandos de propiedad de objeto porque los arrays no tienen objeto asociado.*

Resultado:

France	
Brittany	
Brest	120000
Quimper	80000
Rennes	200000
Normandy	
Caen	75000
Deauville	4000

Líneas ocultas

Cuando todas las líneas de una subjerarquía están ocultas, la línea de ruptura se oculta automáticamente. En el ejemplo anterior, si las líneas 1 a 3 están ocultas, la línea de ruptura "Brittany" no aparecerá.

Gestión de las selecciones y de las posiciones

Puede optimizar la visualización y la gestión de los list box jerárquicos utilizando los eventos formulario On Expand y On Collapse.

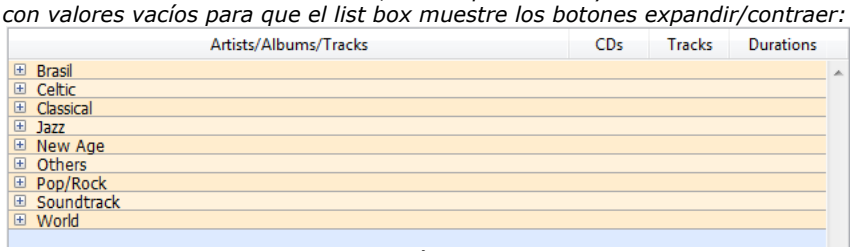
*Un list box jerárquico se construye a partir de los contenidos de sus arrays por lo que sólo se pueden mostrar cuando todos los arrays se cargan en memoria. Esto hace difícil la generación de list box jerárquicos de gran tamaño basados en arrays generados a partir de los datos (vía el comando **SELECTION TO ARRAY**), no sólo por la velocidad de visualización, sino también por el uso de la memoria.*

El uso los eventos formulario On Expand y On Collapse permite superar estas limitaciones: por ejemplo, ahora se puede mostrar sólo una parte de la jerarquía y efectuar la carga y la descarga de los arrays sobre la marcha, en función de las acciones del usuario.

*En el contexto de estos eventos, el comando **LISTBOX GET CELL POSITION** devuelve la celda donde el usuario hizo clic para expandir o contraer una línea.*

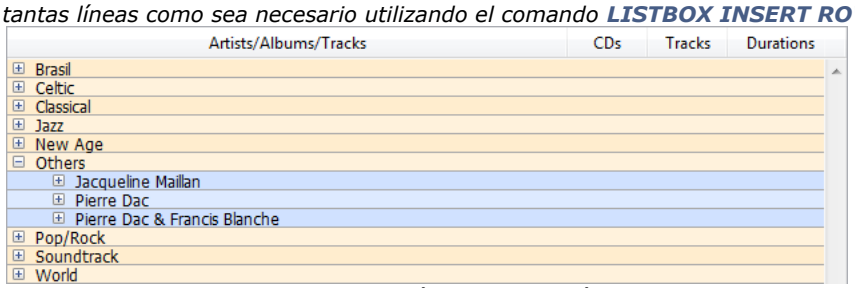
En este caso, debe llenar y vaciar los arrays por código. Los principios a aplicar son:

- *Durante la visualización del list box, sólo el primer array debe llenarse. Sin embargo, usted debe crear un segundo array con valores vacíos para que el list box muestre los botones expandir/contraer:*



- *Cuando un usuario hace clic en un botón de despliegue, puede procesar el evento On Expand. El comando **LISTBOX GET CELL POSITION** devuelve la celda en cuestión y le permite crear la jerarquía apropiada: se llena el primer array con los valores repetidos y el segundo con los valores enviados desde el comando **SELECTION TO ARRAY** e insertar en el list box*

*tantas líneas como sea necesario utilizando el comando **LISTBOX INSERT ROWS**.*

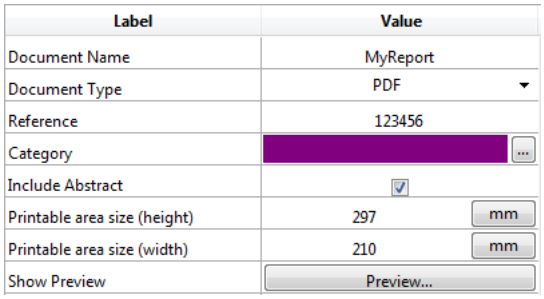


- *Cuando un usuario hace clic en un botón de contracción, puede procesar el evento On Collapse. El comando **LISTBOX GET CELL POSITION** devuelve la celda en cuestión: elimina del list box tantas líneas como sea necesario utilizando el comando **LISTBOX DELETE ROWS**.*

Utilizar arrays objetos en las columnas (4D View Pro)

A partir de 4D v15, las columnas de list box pueden contener arrays de objetos. Como los arrays de objetos pueden contener diferentes tipos de datos, esta nueva y poderosa funcionalidad le permite mezclar diferentes tipos de valores en las líneas de una sola columna, y mostrar varios widgets también. Por ejemplo, puede insertar una entrada de texto en la primera fila, una casilla de selección en la segunda, y una lista desplegable en la tercera. Los arrays de objetos también ofrecen acceso a nuevos tipos de widgets, tales como botones o selectores de colores.

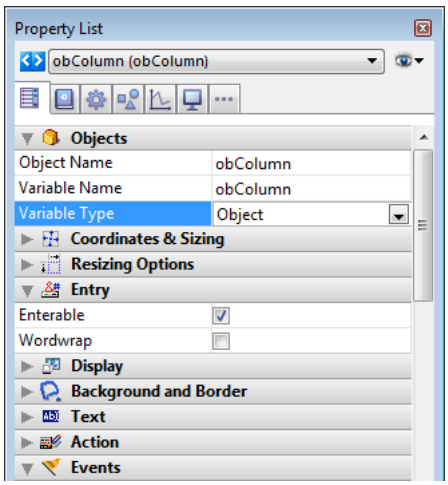
El siguiente list box fue diseñado utilizando un array objeto:



***Nota sobre las licencias**: la capacidad de utilizar arrays de objetos en list boxes es un primer paso para la próxima herramienta "4D View pro" que va a sustituir progresivamente el plug-in 4D View. Esta funcionalidad requiere una licencia 4D View válida. Para más información, por favor consulte el sitio Web de 4D.*

Configurar un columna array de objetos

*Para asignar una array de objetos a una columna de list box, sólo tiene que definir el nombre del array de objetos, ya sea en la lista de propiedades (campo "Nombre de la variable"), o utilizando el comando **LISTBOX INSERT COLUMN** al igual que con toda columna asociada a un array. En la lista de propiedades, ahora puede seleccionar **Objeto** como "Tipo de variable" para la columna:*



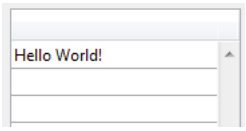
Las propiedades estándar relacionadas con coordenadas, tamaño y estilo están disponibles para las columnas de tipo objeto. Puede definirlas utilizando la lista de propiedades o programando los atributos de estilo, color de fuente, color de fondo y visibilidad para cada línea de columna objeto del list box. Este tipo de columna también se pueden ocultar.

*Sin embargo, el tema **Fuente de datos** no está disponible para las columnas objeto de los list box. De hecho, el contenido de cada celda de la columna se basa en los atributos presentes en el elemento correspondiente del array de objetos. Cada elemento del array puede definir:*

- *el tipo de valor (obligatorio): texto, color, evento, etc.*
- *el valor en sí (opcional): utilizado para la entrada/salida.*
- *la visualización de contenido de la celda (opcional): botón, lista, etc.*
- *ajustes adicionales (opcional): dependen del tipo de valor.*

Para definir estas propiedades, es necesario definir los atributos adecuados en el objeto (los atributos disponibles se listan a continuación). Por ejemplo, puede escribir "Hello World!" en una columna objeto utilizando este simple código:

```
ARRAY OBJECT(obColumn;0) // array de columnas
C_OBJECT($ob) //primer elemento
OB SET($ob;"valueType";"text") //define el tipo de valor (obligatorio)
OB SET($ob;"value";"Hello World!") //define el valor
APPEND TO ARRAY(obColumn;$ob)
```



Nota: *el formato de visualización y los filtros de entrada no se pueden definir para las columnas objeto. Dependen de forma automática del tipo de valor.*

valueType y visualización de los datos

Cuando una columna de list box está asociada a un array de objetos, la visualización, la entrada y la edición de las celdas se basa en el atributo ***valueType*** *presente en cada elemento del array. Los valores* ***valueType*** *soportados son:*

- *"texto": para un valor texto*
- *"real": para un valor numérico que puede incluir separadores como un <espacio>, <.>, o <,>*
- *"integer": para un valor entero*
- *"boolean": para un valor True/False*
- *"color": para definir un color de fondo*
- *"event": para mostrar un botón con una etiqueta.*

4D utiliza los widgets por defecto en función del valor "valueType" (es decir, un "texto" se muestra como un widget de entrada de texto, un "booleano", como una casilla de selección), pero también están disponibles representaciones alternativas vía las opciones (por ejemplo, un real también puede representarse como un menú desplegable). La siguiente tabla muestra la visualización por defecto, así como también las alternativas para cada tipo de valor:

valueType	***Widget por defecto***	***Widgets alternativos***
text	*área de entrada de texto*	*menú desplegable (lista obligatoria) o combo box (lista de opciones)*
real	*área de entrada de texto controlada (números y separadores)*	*menú desplegable (lista obligatoria) o combo box (lista de opciones)*
integer	*controlled text input (numbers only)*	*menú desplegable (lista obligatoria) o combo box (lista de opciones) o casilla de selección de tres estados*
boolean	*casilla de selección*	*menú desplegable (lista obligatoria)*
color	*color de fondo*	*texto*
event	*botón con etiqueta*	
		Todos los widgets pueden tener un botón adicional ***unit toggle button*** *o* ***ellipsis button*** *asociado a la celda.*

Defina la visualización de la celda y las opciones utilizando los atributos específicos en cada objeto (ver abajo).

Formatos de visualización y filtros de entrada

No se puede establecer formatos de visualización o filtros de entrada para columnas objeto de list box. Se definen automáticamente de acuerdo con el tipo de valor. Se enumeran en la siguiente tabla:

Value type	***Formato por defecto***	***Control de entrada***
text	*el mismo que el del objeto*	*sin (no control)*
real	*el mismo que el del objeto (utilización del separador decimal del sistema)*	*"0-9", "." y "-"* *"0-9" y "." si min>=0*
integer	*el mismo que el del objeto*	*"0-9" and "-"* *"0-9" si min>=0*
Boolean	*casilla de selección*	*N/A*
color	*N/A*	*N/A*
event	*N/A*	*N/A*

Atributos

Cada elemento del array objeto de objetos es un objeto que puede contener uno o más atributos que definirán el contenido de la celda y la visualización de datos (ver el ejemplo anterior).

El único atributo obligatorio es "valueType" y sus valores soportados son "text", "real", "integer", "boolean", "color" y "event". La siguiente tabla muestra todos los atributos soportados en arrays de objetos de list box, en función del valor "valueType" (cualquier otro atributo se ignora). Los formatos de visualización y ejemplos se muestran a continuación.

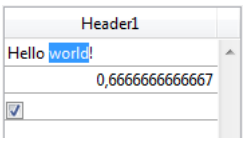
	valueType	text	real	integer	boolean	color	event
Atributos	**Descripción**						
value	valor de la celda (entrada o salida)	x	x	x			
min	valor mínimo		x	x			
max	valor máximo		x	x			
behavior	valor "threeStates"			x			
requiredList	menú desplegable definido en el objeto	x	x	x			
choiceList	combo box definido en objeto	x	x	x			
requiredListReference	RefList 4D, depende del valor de "saveAs"	x	x	x			
requiredListName	nombre de lista 4D, depende del valor "saveAs"	x	x	x			
saveAs	"reference" o "value"	x	x	x			
choiceListReference	RefList 4D, muestra un combo box	x	x	x			
choiceListName	nombre de lista 4D, muestra un combo box	x	x	x			
unitList	array de X elementos	x	x	x			
unitReference	índice del elemento seleccionado	x	x	x			
unitsListReference	RefList 4D para las unidades	x	x	x			
unitsListName	nombre de lista 4D para las unidades	x	x	x			
alternateButton	añadir un botón alternativo	x	x	x	x	x	

value

Los valores de las celdas se almacenan en el atributo "value". Este atributo se utiliza para la entrada y salida. También se puede utilizar para definir los valores por defecto cuando se utilizan las listas (ver abajo).

Ejemplo:

```
ARRAY OBJECT(obColumn;0) //column array
<p>C_OBJECT($ob1)
$entry:="Hello world!"
OB SET($ob1;"valueType";"text")
OB SET($ob1;"value";$entry) // si el usuario introduce un nuevo valor, $entry contendrá el nuevo valor
C_OBJECT($ob2)
OB SET($ob2;"valueType";"real")
OB SET($ob2;"value";2/3)
C_OBJECT($ob3)
OB SET($ob3;"valueType";"boolean")
OB SET($ob3;"value";True)

APPEND TO ARRAY(obColumn;$ob1)
APPEND TO ARRAY(obColumn;$ob2)
APPEND TO ARRAY(obColumn;$ob3)
```



Nota: *el valor Null es soportado y resulta en una celda vacía.*

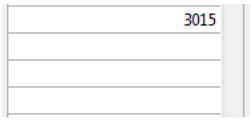
min y max

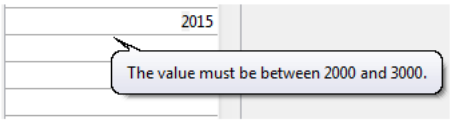
*Cuando "valueType" es "real" o "integer", el objeto acepta también los atributos **min** y **max** con los valores apropiados (los valores deben ser del mismo tipo que el valueType).*

*Estos atributos se pueden utilizar para controlar el rango de valores de entrada. Cuando se valida una celda (cuando pierde el foco), si el valor de entrada es menor que el valor min o mayor que el valor **max**, entonces se rechaza. En este caso, el valor anterior se mantiene y un mensaje de ayuda muestra una explicación.*

Ejemplo:

```
C_OBJECT($ob3)
$entry3:=2015
OB SET($ob3;"valueType";"integer")
OB SET($ob3;"value";$entry3)
OB SET($ob3;"min";2000)
OB SET($ob3;"max";3000)
```





behavior

*El atributo **behavior** ofrece variaciones de la representación estándar de los valores. En 4D v15, una sola variación es posible:*

Atributo	Valor disponible	valueType(s)	Descripción
behavior	threeStates	integer	Representa un valor numérico como casilla de selección de tres estados. 2=semi-marcado, 1=marcado, 0=no marcado, -1=invisible, -2=no marcado desactivado, -3=marcado desactivado, -4=semi-marcado desactivado

Ejemplo:

```
C_OBJECT($ob3)
OB SET($ob3;"valueType";"integer")
OB SET($ob3;"value";-3)
C_OBJECT($ob4)
OB SET($ob4;"valueType";"integer")
OB SET($ob4;"value";-3)
OB SET($ob4;"behavior";"threeStates")
```

requiredList y choiceList

Cuando un atributo "choiceList" o "requiredList" está presente en el interior del objeto, la entrada de texto se sustituye por una lista desplegable o un combo box, dependiendo del atributo:

- *Si el atributo es "choiceList", la celda se muestra como un combo box. Esto significa que el usuario puede seleccionar o escribir un valor.*
- *Si el atributo es "requiredList" entonces la celda se muestra como una lista desplegable. Esto significa que el usuario sólo puede seleccionar uno de los valores de la lista.*

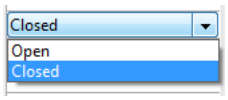
En ambos casos, un atributo "value" se puede utilizar para preseleccionar un valor en el widget.

Nota: *los valores de widgets se definen a través de un array. Si desea asignar una lista 4D existente al widget, es necesario utilizar los atributos "requiredListReference", "requiredListName", "choiceListReference", o "choiceListName".*

Ejemplos:

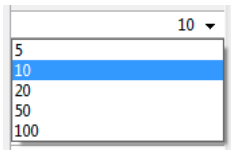
- *Usted quiere mostrar una lista desplegable con sólo dos opciones: "Open" o "Closed". "Closed" debe estar preseleccionada:*

```
ARRAY TEXT($RequiredList;0)
APPEND TO ARRAY($RequiredList;"Open")
APPEND TO ARRAY($RequiredList;"Closed")
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"value";"Closed")
OB SET ARRAY($ob;"requiredList";$RequiredList)
```



- *Usted quiere aceptar todo valor entero, pero mostrar un combo box para sugerir los valores más comunes:*

```
ARRAY LONGINT($ChoiceList;0)
APPEND TO ARRAY($ChoiceList;5)
APPEND TO ARRAY($ChoiceList;10)
APPEND TO ARRAY($ChoiceList;20)
APPEND TO ARRAY($ChoiceList;50)
APPEND TO ARRAY($ChoiceList;100)
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"value";10) //10 as default value
OB SET ARRAY($ob;"choiceList";$ChoiceList)
```



requiredListName y requiredListReference

*Los atributos "requiredListName" y "requiredListReference" le permiten utilizar, en una celda de list box, una lista definida en 4D en modo Diseño (en la caja de herramientas) o por programación (utilizando el comando **New list**). La celda se mostrará como una lista desplegable. Esto significa que el usuario sólo puede seleccionar uno de los valores de la lista.*

Utilice "requiredListName" o "requiredListReference" dependiendo del origen de la lista: si la lista proviene de la caja de herramientas, pase un nombre; de lo contrario, si la lista se ha definido por programación, pase una referencia. En ambos casos, un atributo "valor" se puede utilizar para preseleccionar un valor en el widget.

Nota: *si desea definir estos valores a través de un array simple, es necesario utilizar el atributo "requiredList".*

En este caso, el atributo "saveAs" definirá si el elemento seleccionado debe ser guardado como un "valor" o como una "referencia".

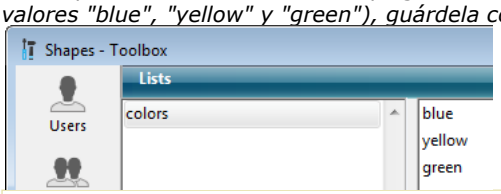
- *Si "saveAs" = "reference" entonces se guardará como una referencia y el "valueType" debe ser real o entero.*
- *Si "saveAs" = "value" se guarda el valor. En este caso, el "valueType" debe ser del mismo tipo que los valores de la lista, "texto"o "entero" por lo general; de lo contrario 4D intentará convertir el valor de la lista en el "valueType" del objeto (ver ejemplos a continuación).*

*Para más información sobre la opción "guardar como", consulte la sección **Guardar como Valor o Referencia** en el Manual de Diseño.*

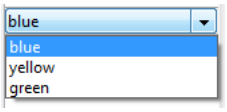
***Nota:** si la lista contiene elementos de texto que representan los valores reales, el separador decimal debe ser un punto ("."), independientemente de los parámetros locales: "17.6" "1234.456".*

Ejemplos:

- *Usted quiere mostrar una lista desplegable basado en una lista "colors" definida en la caja de herramientas (contiene los valores "blue", "yellow" y "green"), guárdela como un valor y muestre "blue" por defecto:*

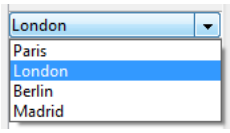



```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"saveAs";"value")
OB SET($ob;"value";"blue")
OB SET($ob;"requiredListName";"colors")
```



- *Usted quiere mostrar una lista desplegable basado en una lista definida por programación y guárdela como una referencia:*

```
<>List:=New list
APPEND TO LIST(<>List;"Paris";1)
APPEND TO LIST(<>List;"London";2)
APPEND TO LIST(<>List;"Berlin";3)
APPEND TO LIST(<>List;"Madrid";4)
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"saveAs";"reference")
OB SET($ob;"value";2) //muestra London por defecto
OB SET($ob;"requiredListReference";<>List)
```



choiceListName y choiceListReference

*Los atributos "choiceListName" y "choiceListReference" le permiten utilizar, en una celda de list box, una lista definida en 4D en modo Diseño (en la caja de herramientas) o por programación (utilizando el comando **New list**). La celda a continuación se muestra como un combo box, lo que significa que el usuario puede seleccionar o digitar un valor.*

Utilice "choiceListName" o "choiceListReference" dependiendo del origen de la lista: si la lista proviene de la caja de herramientas, se pasa un nombre; de lo contrario, si la lista se ha sido definido por programación, se pasa una referencia. En ambos casos, un atributo "value" se puede utilizar para preseleccionar un valor en el widget.

***Nota:** si desea definir estos valores a través de un array simple, es necesario utilizar el atributo "choiceList".*

El atributo "saveAs" no se puede usar en este caso porque los elementos seleccionados se guardan automáticamente como un "value" (ver para más información).

***Nota:** si la lista contiene elementos de texto que representa los valores reales, el separador decimal debe ser un punto ("."), Independientemente de los parámetros locales, por ejemplo: "17.6" "1234.456".*

Ejemplo:

Usted desea mostrar un combo box based on a "colors" basado en una lista "colors", definida en la caja de herramientas (que contiene los valores "blue", "yellow" y "green") y mostrar "green" por defecto:

```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"value";"blue")
OB SET($ob;"choiceListName";"colors")
```

unitsList, unitsListName, unitsListReference y unitReference

Puede utilizar atributos específicos para agregar unidades asociadas a valores de las celdas (por ejemplo: "10 cm", "20 píxeles", etc.). Para definir la lista de unidades, puede utilizar uno de los siguientes atributos:

- *"unitsList": un array que contiene los elementos x utilizados para definir las unidades disponibles (por ejemplo: "cm", "pulgadas", "km", "millas", etc.). Utilice este atributo para definir las unidades dentro del objeto.*
- *"unitsListReference":una referencia a una lista 4D que contiene las unidades disponibles. Utilice este atributo para definir las unidades con una lista 4D creada con el comando **New list**.*
- *"unitsListName": un nombre de una lista 4D creada en modo diseño que contenga las unidades disponibles. Utilice este atributo para definir las unidades con una lista 4D creada en la caja de herramientas.*

Independientemente de la forma en que se defina la lista unidades, puede estar asociada al atributo siguiente:

- *"unitReference": un único valor que contiene el índice (de 1 a x) del elemento seleccionado en la lista de valores "unitList", "unitsListReference" o "unitsListName".*

La unidad actual se muestra como un botón mostrando los valores "unitList", "unitsListReference" o "unitsListName" v cada vez que se hace clic (por ejemplo, "píxeles" -> "filas" -> "cm" -> "píxeles" - > etc.)

***Ejemplo:** queremos definir un valor de entrada numérico seguido de dos unidades posibles: "filas" o "píxeles". El valor actual es "2" + "líneas". Utilizamos valores definidos directamente en el objeto (atributo "unitsList"):*

```
ARRAY TEXT($_units;0)
APPEND TO ARRAY($_units;"lines")
APPEND TO ARRAY($_units;"pixels")
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"value";2) // 2 "units"
OB SET($ob;"unitReference";1) //"lines"
OB SET ARRAY($ob;"unitsList";$_units)
```

alternateButton

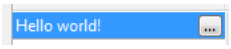
*Si desea agregar un botón de puntos suspensivos [...] a una celda, sólo tiene que pasar el atributo "alternateButton" con el valor **True** en el objeto. El botón se muestra en la celda de forma automática.*

Cuando un usuario clic en este botón, se generará un evento On Alternate Click, y podrá manejarlo como quiera (ver el párrafo "Gestión de eventos" para más información).

***Nota:** On Alternate Click es el nuevo nombre del evento On Arrow Click, renombrado en 4D v15 para resaltar su alcance extendido.*

Ejemplo:

```
C_OBJECT($ob1)
$entry:="Hello world!"
OB SET($ob;"valueType";"text")
OB SET($ob;"alternateButton";True)
OB SET($ob;"value";$entry)
```

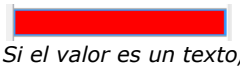


color valueType

El atributo valueType de valor "color" le permite mostrar un color o un texto.

- *Si el valor es un número, un rectángulo de color se dibuja dentro de la celda. Ejemplo:*

```
C_OBJECT($ob4)
OB SET($ob4;"valueType";"color")
OB SET($ob4;"value";0x00FF0000)
```



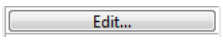
- *Si el valor es un texto, a continuación, se muestra el texto (por ejemplo: "valor", "Automático").*

event valueType

El "evento" valueType muestra un botón simple que genera un evento On Clicked cuando el usuario hace clic. No se pueden pasar o devolver datos o valores.

Opcionalmente, se puede pasar un atributo "label".
Ejemplo:

```
C_OBJECT($ob)
OB SET($ob;"valueType";"event")
OB SET($ob;"label";"Edit...")
```



Gestión de eventos

Varios eventos se pueden manejar en las columnas de list box de tipo array de objetos:

- ***On Data Change**: un evento On Data Change se dispara al modificar un valor:*
  - *en un área de entrada de texto*
  - *en una lista desplegable*
  - *en un área de combo box*
  - *en un botón de unidad (cambio de valor de x a valor x+1)*
  - *en una casilla de selección (cambia entre marcado/no marcado)*
- ***On Clicked**: cuando el usuario hace clic en un botón instalado utilizando el atributo valueType "event", un evento On Clicked se generará. Este evento es manejado por el programador.*
- ***On Alternative Click**:cuando el usuario hace clic en un botón de puntos suspensivos (atributo "alternateButton"), un evento On Alternative Click se generará. Este evento es manejado por el programador.*

***Nota: On Alternative Click** es el nuevo nombre del evento **On Arrow Click** que estaba disponible en versiones anteriores de 4D. Este evento ha sido renombrado en 4D v15 ya que su alcance se ha extendido.*

LISTBOX COLLAPSE

LISTBOX COLLAPSE ({* ;} objeto {; recursivo {; selector {; grueso {; columna}}}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena), Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variables (si se omite *)
recursivo	Booleano	➡	True = contraer los subniveles False = no contraer los subniveles
selector	Entero largo	➡	Parte del list box a contraer
grueso	Entero largo	➡	Número de línea de la ruptura a contraer o Número de nivel del listbox a contraer
columna	Entero largo	➡	Número de columna de la ruptura a contraer

Descripción

El comando **LISTBOX COLLAPSE** se utiliza para contraer líneas de ruptura del objeto list box designado por los parámetros objeto y * parámetros.

Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.

Si el list box no está configurado en modo jerárquico, el comando no hace nada. Para obtener más información acerca de los list boxes jerárquicos, por favor diríjase a la sección **Gestión de list box jerárquicos**.

El parámetro opcional recursivo se utiliza para configurar la contracción de los subniveles jerárquicos del list box. Pase True u omita este parámetro para que el comando contraiga todos los niveles y subniveles. Si pasa False, sólo el primer nivel se contraerá.

El parámetro opcional selector se utiliza para especificar el alcance de la orden. Puede pasar una de las siguientes constantes, que se encuentran en el tema , en este parámetro:

Constante	Tipo	Valor	Comentario
lk all	Entero largo	0	El comando afecta todos los subniveles (valor por defecto, utilizado si el parámetro se omite).
lk selection	Entero largo	1	El comando afecta los subniveles seleccionados.
lk break row	Entero largo	2	El comando afecta el subnivel al que pertenece la "celda" designada por los parámetros línea y columna. Note que estos parámetros representan los números de línea y de columna en el listbox en modo estándar y no en su representación jerárquica. Si los parámetros línea y columna se omiten, el comando no hace nada.
lk level	Entero largo	3	El comando afecta todas las líneas de ruptura correspondientes a la columna nivel. Este parámetro designa un número de columna en el list box en modo estándar y no en su representación jerárquica. Si se omite el parámetro nivel, el comando no hace nada.

Si la selección o el list box no contienen una línea de ruptura o si todas las líneas de ruptura ya están contraídas, el comando no hace nada.

Ejemplo

Este ejemplo colapsa el primer nivel de líneas de ruptura de la selección en el list box:

```
LISTBOX COLLAPSE(*;"MiListbox";False;lk selection)
```

LISTBOX DELETE COLUMN

LISTBOX DELETE COLUMN ({* ;} objeto ; posicionCol {; numero})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
posicionCol	Entero largo	➡	Número de la columna a eliminar
numero	Entero largo	➡	Número de columnas a eliminar

Descripción

El comando ***LISTBOX DELETE COLUMN*** *borra una o más columnas (visibles o no) en el list box designado por los parámetros objeto y *.*

Nota: *este comando no hace nada si se aplica a la primera columna de un list box mostrado en modo jerárquico.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección .*

Si no pasa el parámetro opcional numero, el comando simplemente elimina la columna definida en el parámetro posicionCol. De lo contrario, el parámetro numero indica el número de columnas a eliminar a la derecha comenzando desde la columna posicionCol (esta incluida).

Si el parámetro posicionCol es mayor al número de columnas en el list box, el comando no hace nada.

LISTBOX DELETE ROWS

LISTBOX DELETE ROWS ({* ;} objeto ; posicionL {; numLineas})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
posicionL	Entero largo	➡	Posición de la fila a eliminar
numLineas	Entero largo	➡	Número de líneas a borrar

Descripción

*El comando LISTBOX DELETE ROWS borra una o varias líneas a partir de posicion (visible o no) del list box definido por los parámetros objeto y *.*

***Nota**: este comando funciona únicamente con los list box basados en arrays. Cuando este comando se utiliza con un list box basado en una selección, no hace nada y la variable sistema OK devuelve 0*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.*

Recuerde que después de la ejecución del comando, no habrá ningún elemento seleccionado en el list box.

La fila posicion se elimina automáticamente de todos los arrays utilizados por las columnas del list box.
Si el parámetro posicion es superior al número de líneas del array del list box o si es inferior a 1, el comando no hace nada.

***Nota**: este comando no tiene en cuenta los posibles estados ocultos/visibles de las líneas del list box.*

LISTBOX DUPLICATE COLUMN

LISTBOX DUPLICATE COLUMN ({* ;} objeto ; posCol ; nomCol ; varCol ; nomEncab ; varEncab {; nomPie ; varPie})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Campo o variable (si se omite *)
posCol	Entero largo	➡	Ubicación de la nueva columna duplicada
nomCol	Cadena	➡	Nombre de la nueva columna
varCol	Array, Campo, Variable, Puntero nulo	➡	Nombre de la variable array de la columna o campo o variable
nomEncab	Cadena	➡	Nombre del objeto del encabezado de la columna
varEncab	Variable entera, Puntero nulo	➡	Variable del encabezado de la columna
nomPie	Cadena	➡	Nombre del objeto del pie de la columna
varPie	Variable, Puntero nulo	➡	Variable del pie de la columna

Descripción

*El comando **LISTBOX DUPLICATE COLUMN** duplica la columna definida por los parámetros objeto y * por programación en el contexto del formulario ejecutado (Modo Aplicación). El formulario original, generado en modo Diseño no se modifica.*

***Nota:** esta funcionalidad ya se encuentra en 4D, en modo Diseño únicamente, con el comando **Duplicar Columna** del menú contextual del editor de formularios.*

*De forma predeterminada, todas las opciones de estilo (tamaño, color, formatos, etc) definidos para la columna fuente por medio de la lista de propiedades o mediante los comandos de gestión de objetos (**OBJECT SET COLOR**, etc.) se aplican a la copia. El método objeto y las configuración de los eventos formulario también se duplican.*
Sin embargo, la fuente de datos (array o selección, en función del tipo de fuente definido para el list box), así como los arrays de estilo y de colores no se duplican. Es su responsabilidad definirlos para cada nueva columna después de la duplicación.

*Los parámetros objeto y * designan la columna a duplicar. Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de columna (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable de columna. En este caso, se pasa una referencia de variable en lugar de una cadena.*
***Nota:** este comando no hace nada cuando se aplica a la primera columna de un list box que se muestra en el modo jerárquico.*

La nueva columna duplicada aparece justo antes de la columna designada por el parámetro posicionCol. Si el parámetro posicionCol es mayor que el número total de columnas, a continuación, la columna duplicada se coloca después de la última columna.

En los parámetros nomCol y variableCol, pase el nombre del objeto y de la variable de la nueva columna duplicada.

- *Para list box de tipo array, el nombre de la variable corresponde al nombre del array cuyo contenido se muestra en la columna.*
  Puede pasar un puntero Nil (->[]) en un contexto dinámico (ver abajo).
- *Para list box de tipo selección, puede pasar un campo o una variable en el parámetro variableCol. Así que el contenido de la columna será el valor del campo o de la variable, evaluada para cada registro de la selección asociada al list box. Este tipo de contenido sólo se puede utilizar cuando la propiedad "Fuente de datos" del list box es Selección actual o Selección temporal.*
- *Los list box de tipo colección o selección de entidades no son compatibles con este comando.*

Recuerde que la fuente de datos de la columna original no se duplica: debe establecer una variable, array o campo fuente de la nueva columna duplicada.

En los parámetros nomEncab y variableEncab, pase el nombre del objeto y la variable del encabezado de la nueva columna duplicada. También puede pasar el nombre del objeto y la variable del pie de la columna insertada en los parámetros nomPie y variablePie. Si omite el parámetro variablePie, 4D utiliza una variable dinámica.

***Nota**: los nombres de objetos deben ser únicos en un formulario. Debe asegurarse de que los nombres pasados en nomCol, nomEncab y nomPie no hayan sido utilizados. De lo contrario, la columna no se duplica y se genera un error.*

Este comando debe ser utilizado en el contexto de mostrar un formulario. Se le llama por lo general en el evento de formulario On Load o después de una acción usuario (evento On Clicked).

Duplicación dinámica

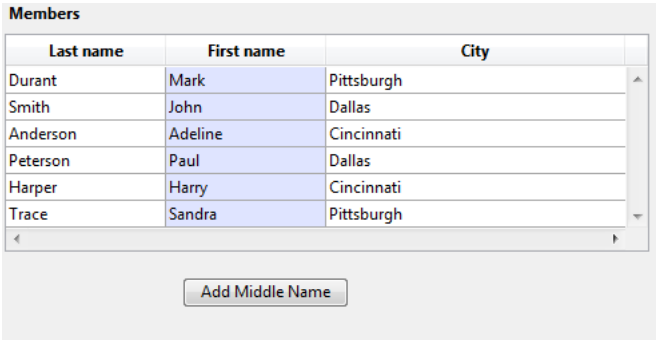
A partir de 4D v14 R3, puede duplicar dinámicamente las columnas de list box y 4D se encarga automáticamente de la definición de las variables necesarias (columna, pie de página y encabezado).

*Para ello, **LISTBOX DUPLICATE COLUMN** acepta un puntero **Nil** (**->[]**) como valor para los parámetros varCol (list box de tipo array únicamente), varEncab y varPie. En este caso, cuando se ejecuta el comando, 4D crea las variables requeridas dinámicamente (para más información, consulte la sección **Variables dinámicas**).*

*Note que las variables de encabezado y de pie de página siempre se crean con un tipo específico (entero largo y texto, respectivamente). Por el contrario, las variables de columna no se pueden escribir durante la creación porque los list boxes aceptan diferentes tipos de arrays para estas variables (array texto, array entero, etc.). Esto significa que usted tiene que fijar el tipo de array manualmente (ver ejemplo 2). Es importante digitar antes de llamar a comandos como **LISTBOX INSERT ROWS** para insertar nuevos elementos en el array. O bien, puede utilizar **APPEND TO ARRAY** tanto para definir el tipo del array y la inserción de elementos.*

Ejemplo 1

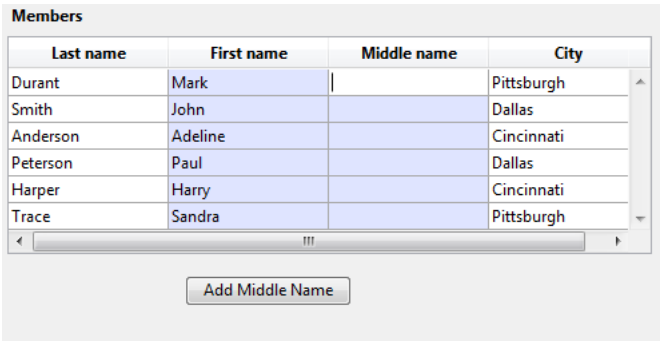
En un list box de tipo array, queremos duplicar la columna "Nombre", lista para la entrada:



Aquí está el código del botón:

```
ARRAY TEXT(arrFirstNames2;Records in table([Members]))
LISTBOX DUPLICATE COLUMN(*;"column2";3;"col2bis";arrFirstNames2;"FirstNameA";vHead2A)
OBJECT SET TITLE(*;"FirstNameA";"Middle Name")
EDIT ITEM(*;"col2A";0)
```

Al hacer clic en el botón, aparece el list box así:



Ejemplo 2

A partir de 4D v14 R3, puede duplicar dinámicamente las columnas de list box y 4D se encarga automáticamente de la definición de las variables necesarias (columna, pie de página y encabezado).

*Para ello, **LISTBOX DUPLICATE COLUMN** acepta un puntero **Nil** (**->[]**) como valor para los parámetros varCol (list box de tipo array únicamente), varEncab y varPie. En este caso, cuando se ejecuta el comando, 4D crea las variables requeridas dinámicamente (para más información, consulte la sección **Variables dinámicas**).*

*Note que las variables de encabezado y de pie de página siempre se crean con un tipo específico (entero largo y texto, respectivamente). Por el contrario, las variables de columna no se pueden escribir durante la creación porque los list boxes aceptan diferentes tipos de arrays para estas variables (array texto, array entero, etc.). Esto significa que usted tiene que fijar el tipo de array manualmente (ver ejemplo 2). Es importante digitar antes de llamar a comandos como **LISTBOX INSERT ROWS** para insertar nuevos elementos en el array. O bien, puede utilizar **APPEND TO ARRAY** tanto para definir el tipo del array y la inserción de elementos.*

LISTBOX EXPAND

LISTBOX EXPAND ({* ;} objeto {; recursivo {; selector {; grueso {; columna}}}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objetos es un nombre de objeto (cadena). Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (si se omite *)
recursivo	Booleano	➡	Verdadero = desplegar los subniveles, Falso = no desplegar los subniveles
selector	Entero largo	➡	Parte del list box a desplegar
grueso	Entero largo	➡	Número de línea de la ruptura a desplegar o Número de nivel del listbox a desplegar
columna	Entero largo	➡	Número de columna de la ruptura a desplegar

Descripción

*El comando **LISTBOX EXPAND** se utiliza para desplegar las líneas de ruptura del objeto list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

*Si el list box no está configurado en modo jerárquico, el comando no hace nada. Para obtener más información sobre los list boxes jerárquicos, consulte la sección **Gestión de list box jerárquicos**.*

El parámetro opcional recursivo se utiliza para configurar el despliegue de los subniveles jerárquicos del list box. Pase Verdadero u omita este parámetro para que el comando provoque el despliegue de todos los niveles y subniveles. Si pasa Falso, sólo el primer nivel especificado se desplegará.

*El parámetro opcional selector se utiliza para especificar el alcance del comando. Puede pasar en este parámetro una de las siguientes constantes del tema **Listbox**:*

Constante	Tipo	Valor	Comentario
lk all	Entero largo	0	El comando afecta todos los subniveles (valor por defecto, utilizado si el parámetro se omite).
lk selection	Entero largo	1	El comando afecta los subniveles seleccionados.
lk break row	Entero largo	2	El comando afecta el subnivel al que pertenece la "celda" designada por los parámetros línea y columna. Note que estos parámetros representan los números de línea y de columna en el listbox en modo estándar y no en su representación jerárquica. Si los parámetros línea y columna se omiten, el comando no hace nada.
lk level	Entero largo	3	El comando afecta todas las líneas de ruptura correspondientes a la columna nivel. Este parámetro designa un número de columna en el list box en modo estándar y no en su representación jerárquica. Si se omite el parámetro nivel, el comando no hace nada.

El comando no selecciona líneas de ruptura.

Si la selección o el list box no contienen una línea de ruptura o si todas las líneas de ruptura ya están expandidas, el comando no hace nada.

Ejemplo

Este ejemplo ilustra diferentes modos de utilizar el comando. Dados los siguientes arrays representados en un list box:

France	Brittany	Brest	120000
France	Brittany	Quimper	80000
France	Brittany	Rennes	200000
France	Normandy	Caen	220000
France	Normandy	Deauville	4000
France	Normandy	Cherbourg	41000
Belgium	Wallonia	Namur	111000
Belgium	Wallonia	Liege	200000
Belgium	Flanders	Antwerp	472000
Belgium	Flanders	Louvain	95000

```
//Desplegar todas las líneas y sublíneas de ruptura del list box
LISTBOX EXPAND(*;"MiListbox")
```

V France	
V Brittany	
Brest	120000
Quimper	80000
Rennes	200000
V Normandy	
Caen	220000
Deauville	4000
Cherbourg	41000
V Belgium	
V Wallonia	
Namur	111000
Liege	200000
V Flanders	
Antwerp	472000
Louvain	95000

```
//Desplegar el primer nivel de líneas de ruptura de la selección
LISTBOX EXPAND(*;"MiListbox";False;lk selection)
//Si la línea "Belgium" no fue seleccionada
```

> France
V Belgium
> Wallonia
> Flanders

```
//Desplegar la línea de ruptura Brittany sin recursividad
LISTBOX EXPAND(*;"MiListbox";False;lk break row;1;2)
```

V France	
V Brittany	
Brest	120000
Quimper	80000
Rennes	200000
> Normandy	
> Belgium	

```
//Desplegar todas las primeras columnas (países) sin recursividad
LISTBOX EXPAND(*;"MiListbox";False;lk level;1)
```

V France
> Brittany
> Normandy
V Belgium
> Wallonia
> Flanders

LISTBOX Get array

LISTBOX Get array ({* ;} objeto ; tipoArray) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Variable (si * se omite)
tipoArray	Entero largo	➡	Tipo de array
Resultado	Puntero	⮌	Puntero al array asociado a la propiedad

Descripción

Nota: este comando sólo funciona con los list box de tipo array.

El comando **LISTBOX Get array** devuelve un puntero al array tipoArray del list box o de la columna de list box designada por los parámetros objeto y * .

Los arrays de estilo, de color, de color de fondo o de control de líneas pueden estar asociados a los list box de tipo de array o (excepto el array de control de líneas) a las columnas de list box array usando la lista de propiedades en modo Diseño o el comando **LISTBOX SET ARRAY**.

Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Puede designar un list box o una columna de list box como parámetro objeto.

Pase en tipoArray, el tipo de array de propiedad a obtener. Puede utilizar una de las siguientes constantes, del tema "**Listbox**":

Constante	Tipo	Valor	Comentario
lk background color array	Entero largo	1	
lk control array	Entero largo	3	
lk font color array	Entero largo	0	
lk row height array	Entero largo	4	(Licencia 4D View Pro requerida)
lk style array	Entero largo	2	

El comando devuelve uno de los siguientes valores:

- **Is nil pointer** si ningún array de la propiedad solicitada está asociado a la columna o al list box.
- un puntero al array de la propiedad solicitada, definido por el usuario.
- un puntero al array de la propiedad solicitada, definido dinámicamente cuando se llama al comando **LISTBOX SET ROW COLOR** o **LISTBOX SET ROW FONT STYLE**

Ejemplo

Ejemplos típicos de uso:

```
vPtr:=LISTBOX Get array(*;"MyLB";lk font color array)
 // devuelve un puntero al array de colores de fuente
 // asociado al list box "MyLB"

vPtr:=LISTBOX Get array(*;"Col4";lk style array)
 // devuelve un puntero al array de estilos de fuente
 // asociado a la columna de list box "Col4"
```

LISTBOX GET ARRAYS

LISTBOX GET ARRAYS ({* ;} objeto ; arrNomsCols ; arrNomsEncabezados ; arrVarCols ; arrVarEncabezados ; arrColsVisibles ; arrEstilos {; arrNomsPies ; arrVarsPies})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
arrNomsCols	Array cadena	⬅	Nombres de objeto de las columnas
arrNomsEncabezados	Array cadena	⬅	Nombres de objeto de los títulos
arrVarCols	Array puntero	⬅	Punteros hacia las variables de las columnas
arrVarEncabezados	Array puntero	⬅	Punteros hacia campos o Nil
arrColsVisibles	Array booleano	⬅	Visibilidad de cada columna
arrEstilos	Array puntero	⬅	Punteros a los arrays o a las variables de estilos de colores y de visibilidad o Nil
arrNomsPies	Array cadena	⬅	Nombres de los objetos de pies de columna
arrVarsPies	Array puntero	⬅	Punteros a las variables de pies de columna

Descripción

*El comando **LISTBOX GET ARRAYS** devuelve un conjunto de arrays sincronizados ofreciendo información sobre cada columna (visible o invisible) del list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.*

Una vez se ejecuta el comando:

- *El array arrNomsCols contiene la lista de los nombres de los objetos para cada columna del list box.*
- *El array arrNomsEncabezados contiene la lista de los nombres de los objetos para cada título de columna del list box.*
- *El array arrVarCols contiene los punteros hacia las variables (arrays) asociadas a cada columna del list box. Para un listbox de tipo selección, arrVarCols contiene:*
  - *Para una columna asociada a un campo, un puntero al campo asociado,*
  - *Para una columna asociada a una variable, un puntero a la variable,*
  - *Para una columna asociada a una expresión, un puntero **Nil**.*
- *El array arrVarEncabezados contiene punteros hacia las variables asociadas a cada título de columna del list box.*
- *El array arrColsVisibles contiene un valor Booleano para cada columna, indicando si la columna es visible (**True**) o oculta (**False**) en el list box.*
- *El array arrEstilos contiene, para un list box de tipo array, cuatro hacia cuatro arrays que permiten aplicar individualmente un estilo, un color de fuente, un color de fondo y un control de visualización personalizado a cada fila del list box. Estos arrays son asociados al list box en la Lista de propiedades del modo Diseño o vía el comando **LISTBOX SET ARRAY**. Si un array no es especificado para el list box, el elemento correspondiente en arrEstilos contendrá un puntero **Nil**.*
  *El cuarto del puntero corresponde ya sea a un array booleano (array de líneas ocultas), o a un array entero largo (array utilizado para definir las líneas ocultas, desactivadas y no seleccionables), en función de la implementación utilizada para el array de control de líneas (ver **Propiedades específicas de los list box**).*
  Para un list box de tipo selección, colección o selección de entidades, arrEstilos contiene:
  - *Por cada configuración definida vía una variable, un puntero a la variable,*
  - *Por cada configuración definida vía una expresión, un puntero **Nil**.*

LISTBOX Get auto row height

LISTBOX Get auto row height ({* ;} objeto ; selector {; unidad}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable.
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (si se omite *)
selector	Entero largo	➡	Valor altura a obtener: lk row min height o lk row max height
unidad	Entero largo	➡	Unidad de valor de altura: 0 = píxeles, 1 = líneas
Resultado	Entero largo	↩	Valor de altura de fila seleccionado

4D View Pro

*Este comando requiere una licencia 4D View Pro. Si esta licencia no está disponible, se muestra un error en el list box cuando el formulario se ejecuta. Para más información, consulte la sección **4D View Pro**.*

Descripción

*El comando **LISTBOX Get auto row height** devuelve el valor de altura de fila mínimo o máximo actual definido para el objeto list box designado utilizando los parámetros objeto y *.*

*El valor actual mínimo o máximo de la altura de la fila puede definirse en la lista de propiedades (ver **Altura de fila automática**) o en el proceso actual utilizando el comando **LISTBOX SET AUTO ROW HEIGHT**.*

***Nota**: este comando solo se puede usar con list boxes no jerárquicos basados en arrays.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Para más información acerca de los nombres de objetos, consulte la sección **Propiedades de los objetos**.*

*En selector, pase el tipo de valor a obtener. Puede utilizar una de las siguientes constantes del tema **Listbox**:*

Constante	Tipo	Valor
lk row max height	Entero largo	33
lk row min height	Entero largo	32

*De forma predeterminada, el comando devuelve el valor en píxeles. Puede pasar una de las siguientes constantes del tema **Listbox** en el parámetro unidad para definir la unidad a utilizar:*

Constante	Tipo	Valor	Comentario
lk lines	Entero largo	1	La altura designa un número de líneas. 4D calcula la altura de una línea en función de la fuente.
lk pixels	Entero largo	0	La altura es un número en píxeles (por defecto)

Ejemplo

Usted desea obtener el número máximo de líneas para una fila de list box:

```
C_LONGINT(vhMax)
vhMax:=LISTBOX Get auto row height(*;"LB";lk row max height;lk lines)
```

LISTBOX GET CELL COORDINATES

LISTBOX GET CELL COORDINATES ({* ;} object ; column ; linea ; izquierda ; arriba ; der ; abajo)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica = objeto es un nombre de objeto (cadena) Si se omite = objeto es una variable
object	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o variable (si * se omite)
column	Entero largo	➡	Número de columna
linea	Entero largo	➡	Número de línea
izquierda	Entero largo	⬅	Coordenada izquierda del objeto
arriba	Entero largo	⬅	Coordenada superior del objeto
der	Entero largo	⬅	Coordenada derecha del objeto
abajo	Entero largo	⬅	Coordenada inferior del objeto

Descripción

*El comando **LISTBOX GET CELL COORDINATES** devuelve en las variables o campos izq, arriba, der y inf las coordenadas (en puntos) de la celda designada por los parámetros columna y linea, en el list box definido por * y objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa el parámetro opcional *, indica que el objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable de tipo de objeto únicamente) en lugar de una cadena.*
*Por coherencia con el comando **OBJECT GET COORDINATES**, el origen es la esquina superior izquierda del formulario que contiene la celda. Además, las coordenadas devueltas son teóricas; que tienen en cuenta el estado de desplazamiento del list box antes de que ocurra cualquier recorte. Como resultado, la celda puede ser no visible (o sólo parcialmente) en sus coordenadas, y estas coordenadas puede estar fuera de los límites del formulario (o incluso negativas). Para saber si la celda es visible (y que parte de ella es visible) es necesario comparar las coordenadas devueltas con las coordenadas del listbox, teniendo en cuenta las siguientes reglas:*

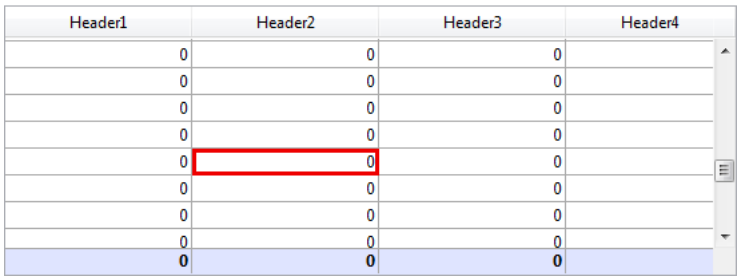
- *Todas las celdas se cortan con las coordenadas de su list box padre (como los devueltos por **OBJECT GET COORDINATES** en el list box).*
- *Los Sub-objetos encabezado y pie de página se muestran sobre el contenido de la columna: cuando las coordenadas de una celda se cruzan las coordenadas de las líneas de encabezado o pie de página, la celda no se muestra en esta intersección.*
- *Los elementos de las columnas bloqueadas se muestran sobre los elementos de las columnas desplazables: cuando las coordenadas de un elemento en una columna desplazable cruzan las coordenadas de una columna bloqueada, entonces que no se muestra en esta intersección.*

*Para más información, consulte la descripción del comando **OBJECT GET COORDINATES**.*

Ejemplo

Usted quiere dibujar un rectángulo rojo alrededor de la celda seleccionada de un listbox:

```
OBJECT SET VISIBLE(*;"RedRect";False) //inicializar un rectángulo rojo
 //el rectángulo ya está definido en alguna parte del formulario
LISTBOX GET CELL POSITION(*;"LB1";$col;$raw)
LISTBOX GET CELL COORDINATES(*;"LB1";$col;$raw;$x1;$y1;$x2;$y2)
OBJECT SET VISIBLE(*;"RedRect";True)
OBJECT SET COORDINATES(*;"RedRect";$x1;$y1;$x2;$y2)
```



LISTBOX GET CELL POSITION

LISTBOX GET CELL POSITION ({* ;} objeto {; X ; Y }; columna ; linea {; varCol})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
X	Real	➡	Coordenada horizontal del ratón
Y	Real	➡	Coordenada vertical del ratón
columna	Entero largo	⬅	Número de columna
linea	Entero largo	⬅	Número de fila
varCol	Puntero	⬅	Puntero a la variable de la columna

Descripción

*El comando **LISTBOX GET CELL POSITION** devuelve los números de la columna y de la fila correspondientes a la ubicación en el list box (designado por * y objeto) del último clic, última selección realizada con el teclado, o las coordenadas vertical y horizontal del ratón.*

Este comando devuelve las coordenadas de un clic o de una acción de selección incluso cuando la entrada de datos no es permitida en el list box.

Si se pasan los parámetros X e Y, este comando devuelve los números de columna y fila correspondientes a las coordenadas del ratón, de lo contrario este comando devuelve los números de columna y fila de un clic o de una acción de selección. El comando devolverá valores válidos incluso cuando la entrada de datos no esté permitida en el list box.

Notas:

- *El valor devuelto en el parámetro linea no tiene en cuenta ningún estado oculto/mostrado de las líneas de list box. También puede devolver un valor de 0 si el clic o la posición Y está por debajo de la última fila.*
- *Si se hace clic en una celda de una columna falsa, o es dado como la posición X, el parámetro columna contiene "N+1", donde N es el número de columnas existentes cuando se hace clic en una celda en una columna falsa o si no hay columna en la posición X. Una columna falsa se puede añadir automáticamente cuando se selecciona la opción "Columna de Autoredimensionamiento"; para más información consulte el párrafo **Tema Opciones de redimensionamiento**).*

El parámetro opcional varCol devuelve un puntero a la variable (es decir al array) asociado con la columna.

Cuando los parámetros X y Y no se utilizan, este comando sólo puede llamarse en el marco de un list box que genere uno de los siguientes eventos:

- *On Clicked y On Double Clicked*
- *On Before Keystroke y On After Keystroke*
- *On After Edit*
- *On Getting Focus y On Losing Focus*
- *On Data Change*
- *On Selection Change*
- *On Before Data Entry*

*Cuando el comando se llama fuera de este contexto, **LISTBOX GET CELL POSITION** devuelve 0 en columna y fila.*

*Este comando tiene en cuenta las acciones de selección o deselección efectuadas con el ratón, con el teclado, o utilizando el comando **EDIT ITEM** (el cual genera el evento On Getting Focus). Si la selección se modifica utilizando las flechas del teclado, columna devuelve 0. En ese caso, el parámetro varCol devuelve **Is nil pointer** si se pasa.*

LISTBOX Get column formula

LISTBOX Get column formula ({* ;} objeto) -> Resultado

Parámetro	*Tipo*		*Descripción*
*	*Operador*	➡	*Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable*
objeto	*Objeto de formulario*	➡	*Nombre de objeto (si se especifica *) o Variable (si se omite *)*
Resultado	*Cadena*	⮌	*Fórmula asociada a la columna*

Descripción

*El comando **LISTBOX Get column formula** devuelve la fórmula asociada a la columna de list box designada por los parámetros objeto y *. Las fórmulas no pueden utilizarse cuando la propiedad "Fuente de datos" del list box es **Selección actual**, **Selección temporal** o **Colección o Selección de entidades**. Si ninguna fórmula está asociada a la columna, el comando devuelve una cadena vacía.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, pase una referencia de variable en lugar de una cadena. Este parámetro debe designar una columna del list box.*

LISTBOX Get column width

LISTBOX Get column width ({* ;} objeto {; anchoMin {; anchoMax}}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
anchoMin	Entero largo	⬅	Ancho mínimo de la columna (en píxeles)
anchoMax	Entero largo	⬅	Ancho máximo de la columna (en píxeles)
Resultado	Entero largo	↩	Ancho de la columna (en píxeles)

Descripción

El comando ***LISTBOX Get column width*** *devuelve el largo (en píxeles) de la columna designada por los parámetros objeto y *. Puede pasar indiferentemente una columna o un título de columna de list box en el parámetro objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección .*

LISTBOX Get column width *puede devolver en los parámetros anchoMin y anchoMax los límites de redimensionamiento de la columna. Estos límites pueden definirse utilizando el comando* ***LISTBOX SET COLUMN WIDTH****.*
Si no se ha definido ningún valor de ancho mínimo y/o máximo para la columna, el parámetro correspondiente devuelve 0.

LISTBOX Get footer calculation

LISTBOX Get footer calculation ({* ;} objeto) -> resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
resultado	Entero largo	⮌	Tipo de cálculo

Descripción

El nuevo comando ***LISTBOX Get footer calculation*** *devuelve el tipo de cálculo asociado al área de pie de página del list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

El parámetro objeto puede designar:

- *la variable o el nombre de un área de pie de página. En este caso, el comando devuelve el cálculo asociado a esta área.*
- *la variable o el nombre de una columna de list box. En este caso, el comando devuelve el cálculo asociado al área de pie de página de esta columna.*

Puede comparar el valor devuelto con las constantes del tema ***List box cálculo pie*** *(ver el comando* ***LISTBOX SET FOOTER CALCULATION****).*

LISTBOX Get footers height

LISTBOX Get footers height ({* ;} objeto {; unidad}) -> resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
unidad	Entero largo	➡	Unidad de valor de altura: 0 o si se omite = píxeles, 1 = líneas
resultado	Entero largo	⮌	Alto de la línea

Descripción

El comando ***LISTBOX Get footers height*** *devuelve la altura de la línea de pie del list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Puede designar indiferentemente el list box o todo pie del list box.*

Por defecto, si omite el parámetro unidad, la altura de línea devuelta se expresa en píxeles. Para definir una unidad diferente, puede pasar una de las siguientes constantes (del tema ***Listbox****), en el parámetro unidad:*

Constante	Tipo	Valor	Comentario
lk lines	Entero largo	1	La altura designa un número de líneas. 4D calcula la altura de una línea en función de la fuente.
lk pixels	Entero largo	0	La altura es un número en píxeles (por defecto)

Nota*: para mayor información sobre el cálculo de la altura de líneas, consulte el manual de Diseño.*

LISTBOX GET GRID

LISTBOX GET GRID ({* ;} objeto ; horizontal ; vertical)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
horizontal	Booleano	⬅	True = visible, False = oculto
vertical	Booleano	⬅	True = visible, False = oculto

Descripción

*El comando **LISTBOX GET GRID** devuelve el estado visible/oculto de las líneas horizontales y/o verticales que componen la rejilla del objeto list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

*El comando devuelve en los parámetros horizontal y vertical, el valor **True** o **False** dependiendo de si las líneas correspondientes se muestran (True) o se ocultan (False). Por defecto, se muestra la rejilla.*

LISTBOX GET GRID COLORS

LISTBOX GET GRID COLORS ({* ;} objeto ; colorH ; colorV)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
colorH	Entero largo	⬅	Valor de color RGB para las líneas horizontales
colorV	Entero largo	⬅	Valor de color RGB para las líneas verticales

Descripción

*El comando **LISTBOX GET GRID COLORS** devuelve el color de las líneas horizontales y verticales que componen la rejilla del objeto list box object designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, pase una referencia de variable en lugar de una cadena.*

*En colorH y colorV, el comando devuelve los valores de los colores RGB. Para mayor información sobre los colores RGB, consulte la descripción del comando **OBJECT SET RGB COLORS**.*

LISTBOX Get headers height

LISTBOX Get headers height ({* ;} objeto {; unidad}) -> resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
unidad	Entero largo	➡	Unidad de valor de altura: 0 o si se omite = píxeles, 1 = líneas
resultado	Entero largo	⮌	Alto de la línea

Descripción

El comando ***LISTBOX Get headers height*** *devuelve la altura de la línea de encabezado del list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Puede designar indiferentemente el list box o todo encabezado del list box.*

Por defecto, si omite el parámetro unidad, la altura de línea devuelta se expresa en píxeles. Para definir una unidad diferente, puede pasar una de las siguientes constantes (del tema ***Listbox****), en el parámetro unidad:*

Constante	Tipo	Valor	Comentario
lk lines	Entero largo	1	La altura designa un número de líneas. 4D calcula la altura de una línea en función de la fuente.
lk pixels	Entero largo	0	La altura es un número en píxeles (por defecto)

Nota*: para mayor información sobre el cálculo de la altura de líneas, consulte el manual de Diseño.*

LISTBOX GET HIERARCHY

LISTBOX GET HIERARCHY ({* ;} objeto ; jerarquico {; jerarquia})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o variables (si * se omite)
jerarquico	Booleano	⬅	True = list box jerárquico, False = list box no jerárquico
jerarquia	Array puntero	⬅	Array de punteros

Descripción

*El comando **LISTBOX GET HIERARCHY** permite buscar las propiedades jerárquicas del objeto list box designado por los parámetros objeto y * .*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

El parámetro booleano jerarquico indica si el listbox está o no en modo jerárquico:

- *Si el parámetro devuelve True, el list box está en modo jerárquico,*
- *Si el parámetro devuelve False, el list box se muestra en modo no jerárquico (modo de array estándar).*

Si el list box está en modo jerárquico, el comando llena el array jerarquia con los punteros a los arrays del list box utilizado para definir la jerarquía.
***Nota**: si el list box está en modo no jerárquico, el comando devuelve, en el primer elemento del array jerarquia, un puntero al array de la primera columna del listbox.*

LISTBOX Get locked columns

LISTBOX Get locked columns ({* ;} objeto) -> resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
resultado	Entero largo	⮌	Número de columnas bloqueadas

Descripción

El comando ***LISTBOX Get locked columns*** *devuelve el número de columnas bloqueadas en el list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

Las columnas pueden desbloquearse vía la Lista de propiedades o con la ayuda del comando ***LISTBOX SET LOCKED COLUMNS****. Para mayor información, consulte el manual de Diseño.*

Si una columna se inserta o se elimina por programación dentro de un área de bloqueo, el número de columnas devueltas por este comando tiene en cuenta este cambio.
Sin embargo, el comando no tiene en cuenta el estado visible/invisible de las columnas.

LISTBOX Get number of columns

LISTBOX Get number of columns ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Entero largo	↩	Número de columnas

Descripción

El comando ***LISTBOX Get number of columns*** *devuelve el número total de columnas (visibles o invisibles) presentes en el list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección* ***Propiedades de los objetos****.*

LISTBOX Get number of rows

LISTBOX Get number of rows ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Entero largo	⬅	Número de filas

Descripción

El comando ***LISTBOX Get number of rows*** *devuelve el número de filas del list box designado por los parámetros objeto y *.*

Nota: LISTBOX Get number of rows *no tiene en cuenta el estado oculto/mostrado de las líneas. Por ejemplo en un list box de 10 líneas, donde las 9 primeras están ocultas,* ***LISTBOX Get number of rows*** *devuelve 10.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección .*

Nota: *si los arrays asociados con las columnas de un list box no tienen todos el mismo tamaño, sólo el número de elementos correspondiente al array más pequeño aparecerá en el list box y de esta manera es devuelto por este comando.*

LISTBOX GET OBJECTS

LISTBOX GET OBJECTS ({* ;} objeto ; arrayNomObjeto)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si * se especifica) o Variable (si * se omite)
arrayNomObjeto	Array texto	⬅	Nombres de los sub objetos del list box (encabezados, columnas, pies)

Descripción

*El comando **LISTBOX GET OBJECTS** devuelve un array que contiene los nombres de todos los objetos que componen el list box designado por los parámetros objeto y * .*

*Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

En arrayNomObjeto, pase un array texto que es llenado automáticamente por el comando. Los nombres de los objetos son devueltos en su orden de presentación, con la siguiente secuencia:

nomCol1
nomEncabCol1
nomPieCol1
nomCol2
nomEncabCol2
nomPieCol2
...

El array devuelve los nombres de los objetos de todas las columnas (incluyendo los pies de columna), independientemente de si son o no visibles.

*Este comando es útil en el contexto del análisis de un formulario utilizando los comandos **FORM LOAD**, **FORM GET OBJECTS** y **OBJECT Get type**. Se puede utilizar, cuando sea necesario, para obtener los nombres de los sub objetos de los list box.*

Ejemplo

Usted quiere cargar un formulario y obtener la lista de todos los objetos de los list boxes que contiene.

```
FORM LOAD("MyForm")
ARRAY TEXT(arrObjects;0)
FORM GET OBJECTS(arrObjects)
ARRAY LONGINT(ar_type;Size of array(arrObjects))
For($i;1;Size of array(arrObjects))
   ar_type{$i}:=OBJECT Get type(*;arrObjects{$i})
   If(ar_type{$i}=Object type listbox)
      ARRAY TEXT(arrLBObjects;0)
      LISTBOX GET OBJECTS(*;arrObjects{$i};arrLBObjects)
   End if
End for
FORM UNLOAD
```

LISTBOX GET PRINT INFORMATION

LISTBOX GET PRINT INFORMATION ({* ;} objeto ; selector ; info)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
selector	Entero largo	➡	Información a obtener
info	Entero largo	⬅	Valor actual

Descripción

El comando ***LISTBOX GET PRINT INFORMATION*** *devuelve la información actual relativa a la impresión del objeto list box designado por los parámetros objeto y * . Este comando permite controlar la impresión del contenido del list box.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

Este comando debe llamarse en el contexto de la impresión de un list box mediante el comando ***Print object****. Fuera de este contexto, no devuelve valores significativos.*

*Pase en selector un valor que indique la información a encontrar y en info una variable de tipo numérico o BLOB. Puede pasar en selector una de las siguientes constantes, del tema "****Listbox****":*

Constante	Tipo	Valor	Comentario
lk last printed row number	Entero largo	0	Devuelve en info el número de la última línea impresa. Permite conocer el número de la próxima línea a imprimir. El número devuelto puede ser mayor al número de líneas efectivamente impresas si el list box contiene las líneas invisibles o si se llama al comando **OBJECT SET SCROLL POSITION** . Por ejemplo, si se han impreso las líneas 1, 18 y 20, info es 20.
lk printed height	Entero largo	3	Devuelve en info la altura en píxeles del objeto efectivamente impreso (incluyendo encabezados, líneas, etc.). Recuerde que si el número de líneas a imprimir es inferior a la "capacidad" del list box, su altura se reduce automáticamente.
lk printed rows	Entero largo	1	Devuelve en info el número de líneas efectivamente impresas durante la última llamada al comando **Print object**. Este número incluye las posibles líneas de ruptura añadidas en el caso de un list box jerárquico. Por ejemplo, info es 10 si el list box contiene 20 líneas y las líneas impares están ocultas.
lk printing is over	Entero largo	2	Devuelve en info un booleano indicando si la última línea (visible) del list box se ha impreso. True = la línea se imprimió; De lo contrario, False.

Para mayor información sobre los principios de impresión de list boxes, consulte ***Impresión de list boxes****.*

Ejemplo 1

Impresión hasta que todas las líneas se impriman:

```
OPEN PRINTING JOB
FORM LOAD("SalesForm")

$Over:=False
Repeat
   $Total:=Print object(*;"mylistbox")
   LISTBOX GET PRINT INFORMATION(*;"mylistbox";lk printing is over;$Over)
   PAGE BREAK
Until($Over)

CLOSE PRINTING JOB
```

Ejemplo 2

Impresión de al menos 500 líneas del list box, conociendo que algunas líneas están ocultas:

```
$GlobalPrinted:=0
Repeat
   $Total:=Print object(*;"mylistbox")
   LISTBOX GET PRINT INFORMATION(*;"mylistbox";lk printed rows;$Printed)
   $GlobalPrinted:=$GlobalPrinted+$Printed
   PAGE BREAK
Until($GlobalPrinted>=500)
```

LISTBOX Get property

LISTBOX Get property ({* ;} objeto ; propiedad) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
propiedad	Entero largo	➡	Información a obtener
Resultado	Cadena, Entero largo	↩	Valor actual

Descripción

*El comando **LISTBOX Get property** devuelve el valor de la propiedad del list box o columna especificado utilizando los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, usted pasa una referencia de variable en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.*

***Nota:** si el list box o columna especificado utilizando los parámetros objeto y * no existe, el comando **LISTBOX Get property** devuelve -1 para los propiedades numéricas o una cadena vacía.*

*En el parámetro propiedad, pase una constante indicando la propiedad cuyo valor quiere obtener. Puede utilizar un valor o una de las siguientes constantes del tema **Listbox**:*

Constante	Tipo	Valor	Comentario
_o_lk display hor scrollbar	Entero largo	2	***Constante obsoleta*** Utilizar el comando **OBJECT GET SCROLLBAR**.
_o_lk display ver scrollbar	Entero largo	4	***Constante obsoleta*** Utilizar el comando **OBJECT GET SCROLLBAR**.
_o_lk footer height	Entero largo	9	***Constante obsoleta*** Utilizar el comando **LISTBOX Get footers height**.
_o_lk header height	Entero largo	1	***Constante obsoleta*** Utilizar el comando **LISTBOX Get headers height**.
_o_lk hor scrollbar position	Entero largo	6	***Constante obsoleta*** Utilizar el comando **OBJECT GET SCROLL POSITION**.
_o_lk ver scrollbar position	Entero largo	7	***Constante obsoleta*** Utilizar el comando **OBJECT GET SCROLL POSITION**.
lk allow wordwrap	Entero largo	14	Propiedad **Wordwrap** Aplica a: Columna* Valores posibles: • lk no (0) • lk yes (1)
lk auto row height	Entero largo	31	Propiedad **Altura de fila automática**. Aplica a: List box o columna Valores posibles: • lk yes • lk no **4D View Pro únicamente:** esta funcionalidad requiere una licencia 4D View Pro. Para más información, consulte **4D View Pro**.
lk background color expression	Cadena	22	**Propiedad Background Color Expression** para list box de tipo selección Aplica a: List box o columna
lk column max width	Entero largo	26	Propiedad **Maximum Width** Aplica a: Columna*
lk column min width	Entero largo	25	
lk column resizable	Entero largo	15	Propiedad **Resizable** Aplica a: Columna* Valores posibles: • lk no (0) • lk yes (1)
lk detail form name	Cadena	19	Propiedad **Detail Form Name** para la selección de tipo list box Aplica a: List box
lk display footer	Entero largo	8	0=oculto, 1=se muestra
lk display header	Entero largo	0	0=oculto, 1=se muestra
lk display type	Entero largo	21	Propiedad **Display Type** para columnas numéricas Aplica a: Columna* Valores posibles: • lk numeric format (0): muestra valores en formato numérico • lk three states checkbox (1): muestra valores como casillas de selección de tres estados
lk double click on row	Entero largo	18	
lk extra rows	Entero largo	13	Propiedad **Hide extra blank rows** Aplica a: List box Valores posibles: • lk display (0) • lk hide (1)
lk font color expression	Cadena	23	Propiedad **Font Color Expression** para list box de tipo selección Aplica a: List box o columna
lk font style expression	Cadena	24	
lk hide selection highlight	Entero largo	16	
lk highlight set	Cadena	27	
lk hor scrollbar height	Entero largo	3	Altura en píxeles

Constante	Tipo	Valor	Comentario
lk multi style	Entero largo	30	Propiedad **Multiestilo** Aplica a: Columna* Valores posibles: • lk no (0)[#/note] • lk yes (1) [#/note]
lk named selection	Cadena	28	Propiedad **Named Selection** para list box de tipo selección Aplica a: List box
lk resizing mode	Entero largo	11	
lk row height unit	Entero largo	17	
lk selection mode	Entero largo	10	Propiedad **Selection Mode** Aplica a: List box Valores posibles: • lk none (0) • lk single (1) • lk multiple (2)
lk single click edit	Entero largo	29	Propiedad **Single-Click Edit** Aplica a: List box Posible valores: • lk no (0) • lk yes (1)
lk sortable	Entero largo	20	**Propiedad** Sortable Aplica a: List box Valores posibles: • lk no (0) • lk yes (1)
lk truncate	Entero largo	12	Propiedad **Truncate with ellipsis** Aplica a: List box o columna Valores posibles: • lk without ellipsis (0) • lk with ellipsis (1)
lk ver scrollbar width	Entero largo	5	Ancho en píxeles

Estas propiedades sólo se aplican a las columnas list box; si pasa un list box como parámetro con una de estas propiedades,* ***LISTBOX Get property* *devuelve -1, o una cadena vacía, dependiendo de la propiedad pasada.*

En general, para indicar un resultado no válido ***LISTBOX Get property*** *devuelve -1 al recuperar las propiedades que tienen valores numéricos, o una cadena vacía; Sin embargo, no se generan errores. Más específicamente, esto ocurre en los siguientes casos:*

- *Si pasa una propiedad que no existe*
- *Si pasa una propiedad que no está disponible para el list box o columna especificada, por ejemplo, usted pasa la propiedad lk font color expression con un list box de tipo array*
- *Si pasa una columna como parámetro con una propiedad que se aplica a un list box, y viceversa, si pasa un list box como parámetro con una propiedad que se aplica a una columna (ver arriba *)*

Además, no es posible devolver valores de más de una columna a la vez; si intenta utilizar el símbolo "@" la parte del nombre de columna para indicar varias columnas con nombres similares, ***LISTBOX Get property*** *devuelve el primer valor coincidente que encuentre; como resultado, el valor devuelto no tiene verdadera importancia.*

Notas:

- *Las constantes lk display footer y lk display header son útiles para calcular el tamaño de un área de list box mostrada en el formulario.*
- *Cuando utilice las constantes lk hor scrollbar position o lk ver scrollbar position, el comando* ***LISTBOX Get property*** *devuelve la posición del cursor de desplazamiento en relación con su posición original, es decir el tamaño de la parte oculta de la ventana, expresado en píxeles. Por defecto, esta posición corresponde a 0. Combinando, por ejemplo, con información relativa a la altura de la fila, este valor le permite encontrar el contenido mostrado en el listbox. Sin embargo, estas constantes son obsoletas y pueden remplazarse por el comando* ***OBJECT GET SCROLL POSITION****.*
- *La instrucción* ***LISTBOX Get property****(vLB; o lk footer height) devuelve el mismo valor que el comando* ***LISTBOX Get footers height*** *cuando los pies se muestran. Sin embargo, si los pies no se muestran,* ***LISTBOX Get property*** *devuelve 0 mientras* ***LISTBOX Get footers height*** *devuelve la altura, en este caso teórica, de los pies.*

Ejemplo 1

Dado un listbox"MyListbox", si ejecuta la siguiente instrucción:

```
$Value:=LISTBOX Get property(*;"MyListbox";lk selection mode) // el valor devuelto indica el modo de selección
```

En este caso, el resultado devuelto indica si varias líneas pueden ser seleccionadas.

Ejemplo 2

Dado un list box "MyListbox", si ejecuta la siguiente instrucción:

```
$resizable:=LISTBOX Get property(*;"MyListbox";lk column resizable)
```

***LISTBOX Get property** devuelve -1 porque la propiedad lk column resizable aplica a columnas y un list box se pasó como parámetro.*

LISTBOX Get row color

LISTBOX Get row color ({* ;} objeto ; fila {; tipoColor}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Variable (si * se omite)
fila	Entero largo	➡	Número de fila
tipoColor	Entero largo	➡	Listbox color de fuente (por defecto) o Listbox color de fondo
Resultado	Entero largo	⬅	Valor de color

Descripción

Nota: *este comando sólo funciona con los list box de tipo array.*

El comando ***LISTBOX Get row color*** *devuelve el color de una fila o de una celda del list box designado por los parámetros objeto y * .*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Puede designar un list box o una columna del list box como parámetro objeto:*

- *si objeto designa un list box, el comando devuelve el color de la fila.*
- *si objeto designa una columna, el comando devuelve el color de la celda.*

En fila, pase el número de la fila cuyo color desea obtener.

Nota: *el comando no tiene en cuenta el posible estado oculto/visible de las filas del list box.*

*En el parámetro tipoColor, puede pasar o la constante lk background color o lk font color (tema "**Listbox**") con el fin de averiguar el color de fondo o el color de la fuente de la fila. Si omite este parámetro, se devuelve el color de la fuente.*

Advertencia: *un color asignado a una fila no se muestra necesariamente en cada celda de la fila (ver el ejemplo). Si los valores de color en conflicto se definen vía las propiedades del list box o de la columna, se aplica un orden de prioridad. Para obtener más información, consulte el manual de Diseño.*

Ejemplo

Dado el siguiente list box:

text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
vColor:=LISTBOX Get row color(*;"Col5";3)
vColor2:=LISTBOX Get row color(*;"List Box";3)
vColor3:=LISTBOX Get row color(*;"List Box";lk background color)
 // vColor contiene 0xFFFF00 (amarillo)
 // vColor2 contiene 0x00FF (azul)
 // vColor3 contiene 0x00FF0000 (rojo)
```

LISTBOX Get row font style

LISTBOX Get row font style ({* ;} objeto ; linea) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto(si se especifica *) o Variable (si se omite *)
linea	Entero largo	➡	Número de línea
Resultado	Entero largo	⮌	Valor de estilo

Descripción

Nota: *este comando sólo funciona con los list box de tipo array.*

El comando ***LISTBOX Get row font style*** *devuelve el estilo de fuente de una línea o de una celda del list box designado por los parámetros objeto y * .*

*Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Puede designar como parámetro objeto un list box o una columna de list box:*

- *si objeto designa un list box, el comando devuelve el estilo de la línea.*
- *si objeto designa una columna, el comando devuelve el estilo de la celda.*

En linea, pase el número de la línea cuyo estilo desea obtener.

Nota: *el comando no tiene en cuenta los estados oculto/visible de las líneas del list box.*

Atención: *un estilo asignado a una línea no se muestra necesariamente en todas las celdas de línea (ver el ejemplo). Si se definen valores de estilo contradictorios usando las propiedades del list box o de la columna, se aplica un orden de prioridad. Para obtener más información, consulte el Manual de Diseño.*

Ejemplo

Dado el siguiente list box:

text	text	text	text	text	text
text	text	text	text	text	text
text	*text*	*text*	*text*	**text**	*text*
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
vStyle:=LISTBOX Get row font style(*;"Col5";3)
vStyle2:=LISTBOX Get row font style(*;"List Box";3)
 // vStyle contiene 1 (Bold)
 // vStyle2 contiene 6 (Italic + Underline)
```

LISTBOX Get row height

LISTBOX Get row height ({* ;} objeto ; fila) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre del objeto (si se especifica *) o Variable (si se omite *)
fila	Entero largo	⇒	Fila de list box, cuya altura desea obtener
Resultado	Entero	⇐	Altura de la fila

4D View Pro

*Este comando requiere una licencia 4D View Pro. Si esta licencia no está disponible, se muestra un error en el list box cuando el formulario se ejecuta. Para más información, consulte la sección **4D View Pro**.*

Descripción

*El comando **LISTBOX Get row height** devuelve la altura de la fila especificada en el objeto list box designado utilizando el objeto y los parámetros *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia variable en lugar de una cadena. Para más información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.*

Si la fila especificada no existe en el list box, el comando devuelve 0 (cero).

*La altura de la fila se devuelve utilizando la unidad definida globalmente por las filas del list box, bien sea en la lista de propiedades o por una llamada antes del comando **LISTBOX SET ROWS HEIGHT**.*

***Nota:** para más información sobre el cálculo de la altura de las filas, consulte el manual de Diseño.*

LISTBOX Get rows height

LISTBOX Get rows height ({* ;} objeto {; unidad}) -> resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
unidad	Entero largo	➡	Unida del valor de altura: 0 o si se omite = píxeles, 1 = líneas
resultado	Entero	↩	Altura de la línea

Descripción

El comando ***LISTBOX Get rows height*** *devuelve la altura actual (en píxeles o en líneas) de las líneas del objeto list box designado utilizando los parámetros objeto y *.El valor devuelto corresponde a la altura de una sola línea.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección* ***Propiedades de los objetos****.*

Por defecto, si omite el parámetro unidad, la altura de línea devuelta se expresa en píxeles. Para definir otra unidad, en el parámetro unidad puede pasar una de las siguientes constantes, del tema ***Listbox****:*

Constante	Tipo	Valor	Comentario
lk lines	Entero largo	1	La altura designa un número de líneas. 4D calcula la altura de una línea en función de la fuente.
lk pixels	Entero largo	0	La altura es un número en píxeles (por defecto)

Nota: *Para mayor información sobre el cálculo de alturas de líneas, consulte el manual de Diseño.*

LISTBOX Get static columns

LISTBOX Get static columns ({* ;} objeto) -> resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
resultado	Entero largo	⮌	Número de columnas estáticas

Descripción

El comando ***LISTBOX Get static columns*** *devuelve el número de columnas estáticas en el list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

Las columnas estáticas pueden definirse vía la Lista de propiedades o con la ayuda del comando ***LISTBOX SET STATIC COLUMNS****.*

Si una columna se inserta o se elimina por programación dentro de un conjunto de columnas estáticas, el número de columnas que devuelve este comando tiene en cuenta este cambio.
Sin embargo, el comando no tiene en cuenta el estado visible/invisible de las columnas.

Nota: *las columnas estáticas y las columnas bloqueadas son dos funciones independientes. Para mayor información, consulte el Manual de Diseño.*

LISTBOX GET TABLE SOURCE

LISTBOX GET TABLE SOURCE ({* ;} objeto ; numTabla {; nombre {; nomSel}})

Parámetro	*Tipo*		*Descripción*
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (si se omite *)
numTabla	Entero largo	⬅	Número de la tabla de la selección
nombre	Cadena	⬅	Nombre de la selección temporal o "" para la selección actual
nomSel	Cadena	⬅	Nombre del conjunto seleccionado

Descripción

*El comando **LISTBOX GET TABLE SOURCE** permite conocer la fuente actual de datos mostrados en el list box designado por los parámetros * y objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, no pase una cadena sino una referencia de variable. Para mayor información sobre nombres de objeto, por favor consulte la sección **Propiedades de los objetos**.*

El comando devuelve en el parámetro numTabla el número de la tabla principal asociada al list box y en el parámetro opcional tempo el nombre de la selección temporal eventualmente utilizada.

Si las líneas del list box están vinculadas con la selección actual de la tabla, el parámetro nombre, si se pasa, devuelve una cadena vacía. Si las líneas del list box están vinculadas con una selección temporal, el parámetro nombre devuelve el nombre de esta selección temporal.

Si el list box está asociado con arrays, numTabla devuelve -1 y tempo, si se pasa, devuelve una cadena vacía.

LISTBOX INSERT COLUMN

LISTBOX INSERT COLUMN ({* ;} objeto ; posicionCol ; nomCol ; variableCol ; nomEncabezado ; varTitulo {; nomPie ; nomVar})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
posicionCol	Entero largo	➡	Ubicación de la columna a insertar
nomCol	Cadena	➡	Nombre del objeto de la columna
variableCol	Array, Campo, Variable, Puntero nulo	➡	Nombre de la variable de la columna
nomEncabezado	Cadena	➡	Nombre del objeto del título de la columna
varTitulo	Variable entera, Puntero nulo	➡	Variable de título de la columna
nomPie	Cadena	➡	Nombre del objeto de pie de la columna
nomVar	Variable, Puntero nulo	➡	Variable de pie de la columna

Descripción

*El comando **LISTBOX INSERT COLUMN** inserta una columna en el list box designado por los parámetros objeto y *.*

***Nota**: este comando no hace nada si se aplica a la primera columna de un lisbox mostrado en modo jerárquico.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.*

La nueva columna se inserta justo en frente de la columna definida por el parámetro posicionCol. Si el parámetro posicionCol es superior al número total de columnas, la columna se añade después de la última columna.

Pase el nombre del objeto y de la variable de la columna insertada en los parámetros nomCol y variableCol.

- *Con un array tipo list box, el nombre de la variable debe coincidir con el nombre del array cuyo contenido se mostrará en la columna.*
  *Puede pasar un puntero **Nil (->[])** si utiliza el comando en un contexto dinámico cuando el formulario se ejecuta (ver abajo).*
- *Con una selección tipo list box, debe pasar un campo o variable en el parámetro variableCol. El contenido de la columna será entonces el valor del campo o de la variable, evaluado para cada registro de la selección asociada al list box. Este tipo de contenidos sólo puede ser utilizado cuando la propiedad "Fuente de datos" del list box es Selección actual o Selección temporal (ver la sección **Gestión programada de los objetos de tipo List box**). Puede utilizar los campos o las variables de tipo cadena, numérico, Fecha, Hora, Imagen y Booleano.*

*En el contexto de list boxes basadas en las selecciones de registros, **LISTBOX INSERT COLUMN** permite insertar elementos simples (campos o variables). Si quiere manipular expresiones más complejas (tales como fórmulas o métodos), debe utilizar el comando **LISTBOX INSERT COLUMN FORMULA**.*

*Los list box de tipo colección o selección de entidades también son soportados, sin embargo, dado que el parámetro variableCol no acepta expresiones, debe utilizar el comando **LISTBOX SET COLUMN FORMULA** para asignar la fuente de datos. Es más preciso utilizar el comando **LISTBOX INSERT COLUMN FORMULA** en este caso.*

***Nota:** no es posible combinar en un mismo list box columnas de tipo array (fuente de datos array) y columnas de tipo campo o variable (fuente de datos selección).*

Pase el nombre del objeto y de la variable del título de la columna insertada en los parámetros nomEncabezado y varEncabezado Parámetros.

Puede pasar en los parámetros nomPie y variablePie, el nombre del objeto y la variable del pie de la columna insertada.

***Nota:** los nombres de los objetos deben ser únicos en un formulario. Debe asegurarse de que los nombres pasados en los parámetros nomCol, nomTitulo y nomPie no hayan sido utilizados. De lo contrario, la columna no sea crea y se genera un error.*

Inserción dinámica

A partir de 4D v14 R3, puede utilizar este comando para insertar columnas en los list box de forma dinámica durante la ejecución del formulario. 4D manejará automáticamente la definición de las variables necesarias (pie de página y encabezado).

*Para ello, **LISTBOX INSERT COLUMN** acepta un puntero **Nil (->[])** como valor valor para los parámetros variableCol (list box de tipo array únicamente) varEncabezado y variablePie. En este caso, cuando se ejecuta el comando, 4D crea las variables requeridas dinámicamente (para más información, consulte la sección **Variables dinámicas**).*

*Note que las variables de encabezado y de pie de página siempre se crean con un tipo específico (respectivamente texto y entero largo). Por el contrario, las variables de columna no se pueden escribir durante la creación porque los list boxes aceptan diferentes tipos de arrays para estas variables (array texto, array entero, etc.). Esto significa que usted tiene que definir el tipo de array manualmente (ver el ejemplo 3). Es importante realizar esta escribiendo antes de llamar a comandos como **LISTBOX INSERT ROWS** para insertar nuevos elementos en el array. O bien, puede utilizar **APPEND TO ARRAY** tanto para definir el tipo del array como para la inserción de elementos.*

Ejemplo 1

Nos gustaría añadir una columna al final del list box:

```
C_LONGINT(NomVarTitulo;$Ultimo;RecNum)
ALL RECORDS([Tabla 1])
```

```
$RecNum:=Records in table([Tabla 1])
ARRAY PICTURE(Imagen;$RecNum)

$Ultimo:=LISTBOX Get number of columns(*;"ListBox1")+1
LISTBOX INSERT COLUMN(*;"ListBox1";$Ultimo;"ColumnImagen";Imagen;"imagenTitulo";NomVarTitulo)
```

Ejemplo 2

Nos gustaría añadir una columna a la derecha del list box y asociarle los valores del campo [Transporte]Tarifas:

```
$Ultimo:=LISTBOX Get number of columns(*;"ListBox1")+1
LISTBOX INSERT COLUMN(*;"ListBox1";$Ultimo;"CampoCol";[Transporte]Tarifas;"nomTitulo";varTitulo)
```

Ejemplo 3

Usted desea insertar una columna de forma dinámica en un array de tipo list box y definir su encabezado:

```
C_POINTER($NilPtr)
LISTBOX INSERT COLUMN(*;"MyListBox";1;"MyNewColumn";$NilPtr;"MyNewHeader";$NilPtr)
ColPtr:=OBJECT Get pointer(Object named;"MyNewColumn")
ARRAY TEXT(ColPtr->;10)
 //Definition of header
headprt:=OBJECT Get pointer(Object named;"MyNewHeader")
OBJECT SET TITLE(headprt->;"Inserted header")
```

Ejemplo 4

You want to add column to a list box of collection type:

```
 //create collection
C_COLLECTION(mycol)
mycol:=New collection(New object("Employee";"John Doe";"JobTitle";"CEO");New object("Employee";"Mary Smith";"JobTitle";"CTO");New object("Employee";"Jane Turner";"JobTitle";"CFO"))
```

The column contents are evaluated for each element of the collection and referenced with the data source expression, This.Employee, as shown below:

To add a column show the job titles:

```
LISTBOX INSERT COLUMN(*;"myListbox";2;"2nd Column";myCol;"2nd Header";header2)
OBJECT SET TITLE(header2;"Title")
LISTBOX SET COLUMN FORMULA(*;"2nd Column";"This.JobTitle";ls text)
```

The column is added to the list box:

LISTBOX INSERT COLUMN FORMULA

LISTBOX INSERT COLUMN FORMULA ({* ;} objeto ; posicionCol ; nomCol ; formula ; tipoDatos ; nomEncabezado ; varEncabezado {; nomPie ; variablePie})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (si * se omite)
posicionCol	Entero largo	➡	Ubicación de la columna a insertar
nomCol	Cadena	➡	Nombre del objeto de la columna
formula	Cadena	➡	Fórmula 4D asociada a la columna
tipoDatos	Entero largo	➡	Tipo de resultado de la fórmula
nomEncabezado	Cadena	➡	Nombre del objeto del encabezado de la columna
varEncabezado	Variable entera, Puntero nulo	➡	Variable del encabezado de la columna
nomPie	Cadena	➡	Nombre del objeto de pie de la columna
variablePie	Variable, Puntero nulo	➡	Variable de pie de columna

Descripción

El comando **LISTBOX INSERT COLUMN FORMULA** inserta una columna en el listbox designado por los parámetros objeto y *. El comando **LISTBOX INSERT COLUMN FORMULA** es similar al comando **LISTBOX INSERT COLUMN** excepto que puede utilizarse para introducir una fórmula como contenido de una columna.

Este tipo de contenido no puede utilizarse cuando la propiedad "Fuente de datos" del list box es **Selección actual**, **Selección temporal** o **Colección o Selección de entidades** (para mayor información, consulte la sección **Gestión programada de los objetos de tipo List box**).

Nota: este comando no hace nada si se aplica a la primera columna de un list box mostrado en modo jerárquico.

Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, no pase una cadena sino una referencia de variable. Para mayor información sobre los nombres de objetos, consulte la sección **Propiedades de los objetos**.

La nueva columna se inserta justo antes de la columna designada por el parámetro posicionCol. Si el parámetro posicionCol es superior al número total de columnas, la columna se añadirá después de la última columna.

Pase en el parámetro nomCol el nombre del objeto de la columna insertada.

El parámetro formula puede contener cualquier expresión válida:

- Una instrucción,
- Una fórmula generada utilizando el editor de fórmulas,
- Una llamada a un comando 4D,
- Una llamada a un método proyecto.

En el momento de la llamada del comando, la formula se analiza y luego se ejecuta.

Nota: utilice el comando **Command name** para definir las fórmulas independientes del lenguaje de la aplicación (cuando llaman a comandos 4D).

El parámetro tipoDatos puede utilizarse para designar el tipo de datos resultantes de la ejecución de la formula. Debe pasar en este parámetro una de las siguientes constantes del tema **Tipos de campos y variables**:

Constante	Tipo	Valor
Is Boolean	Entero largo	6
Is date	Entero largo	4
Is picture	Entero largo	3
Is real	Entero largo	1
Is text	Entero largo	2
Is time	Entero largo	11

Si el resultado de la formula no corresponde al tipo de datos esperado, se genera un error.

Pase en los parámetros nomEncabezado y varEncabezado el nombre de objeto y la variable del encabezado de la columna insertada.

También puede pasar en los parámetros nomPie y variablePie el nombre del objeto y la variable del pie de la columna insertada. Si omite el parámetro variablePie, 4D utilizará una variable dinámica.

Nota: los nombres de objeto deben ser únicos en un formulario. Debe asegurarse de que los nombres pasados en los parámetros nomCol, nomEncabezado y nomPie no hayan sido utilizados. De lo contrario, la columna no se crea y se genera un error.

Inserción dinámica

A partir de 4D v14 R3, puede utilizar este comando para insertar columnas en los list box de forma dinámica durante la ejecución del formulario. 4D manejará automáticamente la definición de las variables necesarias (pie de página y encabezado).

Para ello, **LISTBOX INSERT COLUMN FORMULA** acepta un puntero **Nil (->[])** como valor para los parámetros varEncabezado y variablePie. En este caso, cuando se ejecuta el comando, 4D crea las variables requeridas dinámicamente (para más información, consulte la sección **Variables dinámicas**).

Note que las variables de encabezado y de pie de página siempre se crean con un tipo específico (entero largo y texto, respectivamente).

Ejemplo 1

Queremos añadir una nueva columna a la derecha del listbox que contendrá una fórmula que calcula la edad de un empleado:

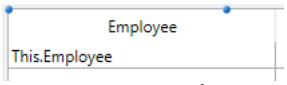
```
vEdad:="Fecha actual-[Empleados]FechaNacimiento)\365"
$ultima:=LISTBOX Get number of columns(*;"ListBox1")+1
LISTBOX INSERT COLUMN FORMULA(*;"ListBox1";$ultima;"ColFormula";vEdad;ls real;"Edad";varEncabezado)
```

Ejemplo 2

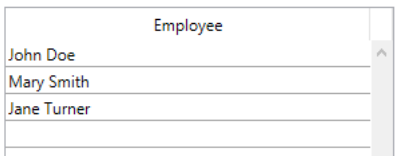
Usted quiere añadir una columna a un list box de tipo colección:

```
 //crear colección
C_COLLECTION(emps)
emps:=New collection(New object("Employee";"John Doe";"JobTitle";"CEO");New object("Employee";"Mary Smith";"JobTitle";"CTO");New object("Employee";"Jane Turner";"JobTitle";"CFO"))
```

El contenido de la columna será evaluado para cada elemento de la colección y se utiliza la expresión fuente This.Employee:



Durante la ejecución:



Para añadir una columna mostrando los títulos de los cargos ocupados:

```
LISTBOX INSERT COLUMN FORMULA(*;"EmpLB";2;"2nd Column";"This.JobTitle";ls text;"JTHeader";header2)
OBJECT SET TITLE(header2;"Title")
```

La columna se añade al list box:

Employee	Title
John Doe	CEO
Mary Smith	CTO
Jane Turner	CFO

LISTBOX INSERT ROWS

LISTBOX INSERT ROWS ({* ;} objeto ; posicionL {; numLineas})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
posicionL	Entero largo	➡	Posición de la fila a insertar
numLineas	Entero largo	➡	Número de líneas a insertar

Descripción

*El comando **LISTBOX INSERT ROWS** inserta una o varias nuevas líneas en el list box designado por los parámetros objeto y *.*

***Nota**: este comando funciona únicamente con los list box basados en arrays. Cuando este comando se utiliza con un list box basado en una selección, no hace nada y la variable sistema OK toma el valor 0.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.*

Por defecto, si se omite el parámetro numLineas, sólo se inserta una línea. De lo contrario, el comando inserta el número de líneas definido en este parámetro.

La fila se inserta en la posición definida por el parámetro posicionL. Una nueva fila se añade automáticamente en esta posición en todos los arrays utilizados por las columnas del list box, cualquiera que sea su tipo y visibilidad.

Si el valor de posicionL es mayor que el número total de filas en el list box, la fila se añade al final de cada array. Si es igual a 0, la fila se añade al principio de cada array. Si contiene un valor negativo, el comando no hace nada.

LISTBOX MOVE COLUMN

LISTBOX MOVE COLUMN ({* ;} objeto ; posicionCol)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (si se omite *) de la columna a mover
posicionCol	Entero largo	➡	Nueva ubicación de la columna

Descripción

*El comando **LISTBOX MOVE COLUMN** mueve por programación la columna designada por los parámetros objeto y * en el contexto del formulario en ejecución (modo Aplicación). El formulario original, generado en modo Diseño, no se modifica.*

*Los parámetros objeto y * designan la columna a mover. Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de columna (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable de columna. En este caso, pase una referencia de variable en lugar de una cadena.*

La columna se mueve justo en frente de la designada por el parámetro posicionCol. Si el parámetro posicionCol es mayor al número total de columnas, luego la columna se mueve hasta justo después de la última columna.

***Nota**: este comando no hace nada cuando se aplica a la primera columna de un list box que se muestra en el modo jerárquico.*

El comando en cuenta las propiedades de las columnas estáticas y bloqueadas: por ejemplo, si intenta mover una columna estática, el comando no hace nada.

Esta funcionalidad está presente en 4D en modo Aplicación: el usuario puede mover las columnas no estáticas utilizando el ratón. Sin embargo, a diferencia del desplazamiento efectuado por el usuario, este comando no genera el evento On Column Moved.

Ejemplo

Usted quiere invertir la segunda y tercera columna del list box:

```
LISTBOX MOVE COLUMN(*;"column2";3)
```

LISTBOX MOVED COLUMN NUMBER

LISTBOX MOVED COLUMN NUMBER ({* ;} objeto ; antPosicion ; nuevPosicion)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
antPosicion	Entero largo	⬅	Posición anterior de la columna movida
nuevPosicion	Entero largo	⬅	Nueva posición de la columna movida

Descripción

*El comando **LISTBOX MOVED COLUMN NUMBER** devuelve dos números en antPosicion y nuevPosicion indicando respectivamente la posición anterior y la nueva posición de la columna movida en el list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección .*

*Este comando debe utilizarse con el evento de formulario **On column moved** (ver el comando **Form event**).*

***Nota:** este comando tiene en cuenta las columnas invisibles.*

LISTBOX MOVED ROW NUMBER

LISTBOX MOVED ROW NUMBER ({* ;} objeto ; antPosicion ; nuevPosicion)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
antPosicion	Entero largo	⬅	Posición anterior de la fila movida
nuevPosicion	Entero largo	⬅	Nueva posición de la fila movida

Descripción

El comando ***LISTBOX MOVED ROW NUMBER*** *devuelve dos números en antPosicion y nuevPosicion indicando respectivamente la posición anterior y la nueva posición de la fila movida en el list box, especificadas por los parámetros objeto y *.*

Nota: *sólo puede mover las líneas en los list box de tipo array.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección* ***Propiedades de los objetos****.*

Este comando debe utilizarse con el evento de formulario On row moved (ver el comando ***Evento formulario*** *).*

Nota: *este comando no tiene en cuenta el estado oculto/mostrado de las líneas del list box.*

LISTBOX SELECT BREAK

LISTBOX SELECT BREAK ({* ;} objeto ; linea ; columna {; accion})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o variable (si se omite *)
linea	Entero largo	➡	Número de línea de la ruptura
columna	Entero largo	➡	Número de columna de la ruptura
accion	Entero largo	➡	Acción de selección

Descripción

*El comando **LISTBOX SELECT BREAK** se utiliza para seleccionar líneas de ruptura en el objeto list box designado por los parámetros objeto y *. El list box debe mostrarse en modo jerárquico.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

*Las líneas de ruptura se añaden para representar la jerarquía, pero no corresponden a las líneas existentes en el array. Para designar una línea de ruptura a seleccionar, debe pasar en los parámetros línea y columna el número de línea y de columna correspondiente a la primera ocurrencia en el array correspondiente. Estos valores son devueltos por el comando **LISTBOX GET CELL POSITION** cuando el usuario ha seleccionado una línea de ruptura. Este principio se describe en el párrafo "Gestión de las líneas de ruptura" de la sección **Gestión de list box jerárquicos**.*

*El parámetro accion, si se pasa, puede definir la acción de selección que debe efectuarse cuando una selección de líneas de ruptura existe en el list box. Puede pasar un valor o una de las siguientes constantes, que se encuentra en el tema "**Listbox**" el tema:*

Constante	Tipo	Valor	Comentario
lk add to selection	Entero largo	1	La línea seleccionada se añade a la selección existente. Si la línea seleccionada ya pertenece a la selección existente, el comando no hace nada.
lk remove from selection	Entero largo	2	La línea seleccionada se remueve de la selección existente. Si la línea especificada no pertenece a la selección existente, el comando no hace nada.
lk replace selection	Entero largo	0	La línea seleccionada se convierte en la nueva selección, reemplazando la selección existente. El comando tiene el mismo efecto que un clic de usuario en una línea (sin embargo, el evento On Clicked no se genera). Esta es la acción por defecto (si se omite el parámetro acción).

***Nota:** si ha seleccionado la opción **Ocultar el resaltado de selección** para un:*

- *tiene que hacer selecciones de list box visibles utilizando opciones de interfaz disponibles. Para más información acerca de cómo hacerlo, ver **Personalizar la apariencia de las selecciones**.*
- *no puede resaltar las líneas de ruptura para los list boxes jerárquicos en este caso (ver **Limitación con list boxes jerárquicos**).*

Ejemplo

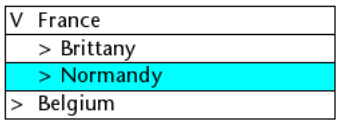
Dados los siguientes arrays representados en un list box:

(T1)	(T2)	(T3)	(T4)
France	Brittany	Brest	120000
France	Brittany	Quimper	80000
France	Brittany	Rennes	200000
France	Normandy	Caen	220000
France	Normandy	Deauville	4000
France	Normandy	Cherbourg	41000
Belgium	Wallonia	Namur	111000
Belgium	Wallonia	Liege	200000
Belgium	Flanders	Antwerp	472000
Belgium	Flanders	Louvain	95000

Queremos seleccionar la línea de ruptura "Normandy" en la representación jerárquica de estos arrays:

```
$row:=Find in array(T2;"Normandy")
$column:=2
LISTBOX COLLAPSE(*;"MyListbox") //contracción de todos los niveles
LISTBOX SELECT BREAK(*;"MyListbox";$row;$column)
```

Este es el resultado:



LISTBOX SELECT ROW

LISTBOX SELECT ROW ({* ;} objeto ; posicionL {; accion})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
posicionL	Entero largo	➡	Número de la fila a seleccionar
accion	Entero largo	➡	Acción de selección

Descripción

El comando ***LISTBOX SELECT ROW*** *selecciona la fila cuyo número se pasa en posicion en el list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección* ***Propiedades de los objetos****.*

El parámetro opcional accion, si se pasa, se utiliza para definir la acción de selección a ejecutar cuando una selección de filas ya existe en el list box. Puede pasar un valor o una de las siguientes constantes (ubicadas en el tema "[#title id="274"/]"):

Constante	Tipo	Valor	Comentario
lk add to selection	Entero largo	1	La línea seleccionada se añade a la selección existente. Si la línea seleccionada ya pertenece a la selección existente, el comando no hace nada.
lk remove from selection	Entero largo	2	La línea seleccionada se remueve de la selección existente. Si la línea especificada no pertenece a la selección existente, el comando no hace nada.
lk replace selection	Entero largo	0	La línea seleccionada se convierte en la nueva selección, reemplazando la selección existente. El comando tiene el mismo efecto que un clic de usuario en una línea (sin embargo, el evento On Clicked no se genera). Esta es la acción por defecto (si se omite el parámetro acción).

Cuando el parámetro posicion no corresponde exactamente a un número de fila existente, el comando actúa de la siguiente manera:

- *Si posicion es <0, el comando no hace nada, cualquiera que sea el valor del parámetro accion.*
- *Si posicion es 0 y el parámetro accion contiene lk replace selection o se omite, todas las filas del list box son seleccionadas. Si el parámetro accion contiene lk remove from selection, todas las filas del list box son deseleccionadas.*
- *Si el valor de posicion es superior al número total de filas contenidas en el list box (sólo en el caso de un array de tipo listbox), el array booleano asociado a el list box es redimensionado automáticamente y la acción de selección se efectúa. Este mecanismo permite utilizar* ***LISTBOX SELECT ROW*** *con los comandos "estándar" de gestión de arrays (tales como* ***APPEND TO ARRAY****) que no provocan la sincronización inmediata del listbox.*

Después de la ejecución del método, los arrays son sincronizados: si el array fuente del listbox ha sido redimensionado efectivamente, la acción de selección se lleva a cabo. De lo contrario, el array booleano asociado con el list box vuelve a su tamaño inicial y el comando no hace nada.

Notas:

- *Si quiere que el list box se desplace automáticamente para mostrar la fila seleccionada, utilice el comando* ***OBJECT SET SCROLL POSITION*** *.*
- *Para pasar una fila a modo edición (para permitir la entrada de datos), utilice el comando* ***EDIT ITEM****.*
- *Si el número pasado en posicion corresponde a una línea oculta en el listbox, la línea es seleccionada pero no se muestra.*
- *Si ha seleccionado la opción* ***Ocultar el resaltado de selección*** *para un list box, deberá hacer las selecciones de list box visibles utilizando las opciones de interfaz disponibles. Para mayor información sobre cómo hacer esto, ver* ***Personalizar la apariencia de las selecciones****.*

LISTBOX SET ARRAY

LISTBOX SET ARRAY ({* ;} objeto ; tipoArray ; ptrArray)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (si se omite *)
tipoArray	Entero largo	➡	Tipo de array
ptrArray	Puntero	➡	Array a asociar a la propiedad

Descripción

***Nota**: este comando sólo funciona con los list box de tipo array.*

*El comando **LISTBOX SET ARRAY** asocia un array de tipo tipoArray al list box o a la columna de list box designada por los parámetros objeto y * .*

***Nota**: los arrays de estilo, de colores o de color de fondo o de control de líneas también pueden estar asociados a los list box de tipo array utilizando la lista de propiedades en modo Diseño.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Puede designar como parámetro objeto un list box o una columna de list box.*

*En tipoArray, pase el tipo de array a asociar al list box o a la columna. Puede utilizar una de las siguientes constantes del tema "**Listbox**":*

Constante	Tipo	Valor	Comentario
lk background color array	Entero largo	1	
lk control array	Entero largo	3	
lk font color array	Entero largo	0	
lk row height array	Entero largo	4	(Licencia 4D View Pro requerida)
lk style array	Entero largo	2	

En el parámetro ptrArray, se pasa un puntero al array a utilizar para soportar el tipo de propiedad.

Ejemplo 1

Usted quiere volver a utilizar el array de colores de fuente de la columna 4ta para la columna 10ma:

```
 // recuperar un puntero al array de la columna 4
$Pointer:=LISTBOX Get array(*;"Col4";lk font color array)
 // verificar que existe
If(Not(Nil($Pointer)))
 //transferir a la columna 10
   LISTBOX SET ARRAY(*;"Col10";lk font color array;$Pointer)
End if
```

Ejemplo 2

Usted desea definir un array de altura de línea para un list box:

```
LISTBOX SET ARRAY(*;"LB";lk row height array;->RowHeightArray)
```

***Nota**: la propiedad **Row Height Array** para list boxes requiere una licencia 4D View Pro. Para más información, consulte **4D View Pro**.*

LISTBOX SET AUTO ROW HEIGHT

LISTBOX SET AUTO ROW HEIGHT ({* ;} objeto ; selector ; valor ; unidad)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable.
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (si se omite)
selector	Entero largo	➡	Valor de la altura a definir: lk row min height o lk row max height
valor	Entero largo	➡	Valor mínimo o máximo de la altura de la fila
unidad	Entero largo	➡	Unidad de valor de altura: 0 = píxeles, 1 = líneas

4D View Pro

*Este comando requiere una licencia 4D View Pro. Si esta licencia no está disponible, se muestra un error en el list box cuando el formulario se ejecuta. Para más información, consulte la sección **4D View Pro**.*

Descripción

*El comando **LISTBOX SET AUTO ROW HEIGHT** le permite establecer el valor de altura de fila mínimo o máximo en el objeto de list box designado utilizando los parámetros objeto y *.*

***Nota**: este comando solo se tiene en cuenta si el list box está denido en modo de altura de fila automática (ver **Altura de fila automática**), lo cual está disponible para listboxes basados en arrays, no jerárquicos. De lo contrario, no tiene ningún efecto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Para más información acerca de los nombres de objetos, consulte la sección **Propiedades de los objetos**.*

*En selector, pase el tipo de valor a definir. Puede utilizar una de las siguientes constantes del tema **Listbox**:*

Constante	Tipo	Valor
lk row max height	Entero largo	33
lk row min height	Entero largo	32

En valor, pase el valor correspondiente en la unidad apropiada.

*El parámetro unidad puede ajustarse utilizando una de las siguientes constantes del tema **Listbox**:*

Constante	Tipo	Valor	Comentario
lk lines	Entero largo	1	La altura designa un número de líneas. 4D calcula la altura de una línea en función de la fuente.
lk pixels	Entero largo	0	La altura es un número en píxeles (por defecto)

***Nota:** el comando no comprueba la consistencia de los valores. Sin embargo, en tiempo de ejecución, el valor mínimo se aplicará a ambos valores en caso de conflicto. Por ejemplo, si el valor mínimo es 5 líneas y el valor máximo es de 3 líneas (que es inconsistente), la altura máxima aplicada a las filas del list box será de 5 líneas.*

Ejemplo

Usted desea definir las alturas mínimas y máximas para un list box con una altura de fila automática:

```
LISTBOX SET AUTO ROW HEIGHT(*;"LB";lk row min height;60;lk pixels) // 60 píxeles para el valor mínimo
LISTBOX SET AUTO ROW HEIGHT(*;"LB";lk row max height;100;lk pixels) //y 100 píxeles para el valor máximo
```

LISTBOX SET COLUMN FORMULA

LISTBOX SET COLUMN FORMULA ({* ;} objeto ; formula ; tipoDato)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
formula	Cadena	➡	Fórmula 4D asociada a la columna
tipoDato	Entero largo	➡	Tipo de resultado de la fórmula

Descripción

El comando ***LISTBOX SET COLUMN FORMULA*** *modifica la formula asociada a la columna de list box designada por los parámetros objeto y *. Las fórmulas no se pueden utilizar cuando la propiedad "Fuente de datos" del list box es* ***Selección actual***, ***Selección temporal*** *o* ***Colección*** *o* ***Selección de entidades***.

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Este parámetro debe designar una columna del list box.*

El parámetro formula puede contener toda expresión valida:

- *una instrucción,*
- *una fórmula generada utilizando el editor de fórmulas,*
- *una llamada a un comando 4D,*
- *una llamada a un método de proyecto.*

Cuando se llama el comando, la fórmula se analiza y luego se ejecuta.

Nota*: utilice el comando* ***Command name*** *para definir las fórmula independientes del lenguaje de la aplicación (cuando se llaman los comandos 4D).*

El parámetro tipoDatos designa el tipo de datos resultantes de la ejecución de la fórmula. En este parámetro, pase una de las constantes del tema ***Tipos de campos y variables****. Si el resultado de la fórmula no corresponde al tipo de datos esperado, se genera un error.*

LISTBOX SET COLUMN WIDTH

LISTBOX SET COLUMN WIDTH ({* ;} objeto ; ancho {; anchoMin {; anchoMax}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
ancho	Entero largo	➡	Ancho de la columna (en píxeles)
anchoMin	Entero largo	➡	Ancho mínimo de columna (en píxeles)
anchoMax	Entero largo	➡	Ancho máximo de columna (en píxeles)

Descripción

*El comando **LISTBOX SET COLUMN WIDTH** le permite modificar por programación el ancho de una o todas las columnas del objeto (list box, columna o título) designado utilizando los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección .*

Pase en el parámetro ancho el nuevo ancho (en píxeles) del objeto.
• Si objeto designa el objeto list box, todas las columnas del list box son redimensionadas.
• Si objeto designa una columna o un título de columna, sólo la columna designada es redimensionada.

Los parámetros opcionales anchoMin y anchoMax permiten definir los límites para el redimensionamiento manual de la columna. Puede pasar en anchoMin y anchoMax respectivamente los valores del ancho mínimo y máximo, expresado en píxeles. Si quiere que el usuario no pueda redimensionar la columna, debe pasar el mismo valor en ancho, anchoMin y anchoMax.

LISTBOX SET FOOTER CALCULATION

LISTBOX SET FOOTER CALCULATION ({* ;} objeto ; calculo)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
calculo	Entero largo	➡	Cálculo para el área de pie

Descripción

El comando ***LISTBOX SET FOOTER CALCULATION*** *permite definir el cálculo automático asociado al área de pie del list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*
El parámetro objeto puede designar:

- *la variable o el nombre de un área de pie de página. En este caso, el comando aplica a esta área.*
- *la variable o el nombre de una columna de list box. En este caso, el comando aplica al área de pie de esta columna.*
- *la variable o el nombre de un list box. En este caso, el comando aplica a las áreas de pie del listbox.*

En el parámetro calculo, pase una de las siguientes constantes del tema ***List box cálculo pie*** *, con el fin de definir el cálculo a efectuar:*

Constante	Tipo	Valor	Comentario
Listbox footer std deviation	Entero largo	7	Utilizable con las columnas de tipo numérico u hora (list boxes de tipo array únicamente) Tipo de resultado por defecto: Real
lk footer average	Entero largo	6	Utilizable con las columnas de tipo numérico, hora Tipo de resultado por defecto:Real
lk footer count	Entero largo	5	Utilizable con las columnas de tipo numérico, texto, fecha, hora, booleano, imagen Tipo de resultado por defecto: Entero largo
lk footer custom	Entero largo	1	Ningún cálculo es efectuado por 4D. La variable del pie debe calcularse por programación. Típo por defecto del resultado del cálculo: tipo de la variable
lk footer max	Entero largo	3	Utilizable con las columnas de tipo numérico, fecha, hora, booleano Tipo de resultado por defecto: tipo del array o campo de la columna
lk footer min	Entero largo	2	Utilizable con las columnas de tipo numérico, fecha, hora, booleano Tipo por defecto del resultado: tipo del array o campo de la columna
lk footer sum	Entero largo	4	Utilizable con las columnas de tipo numérico, hora, booleano Tipo de resultado por defecto: tipo del array o campo de la columna
lk footer sum squares	Entero largo	9	Utilizable con las columnas de tipo numérico, hora (listbox de tipo array únicamente) Tipo por defecto del resultado: Real
lk footer variance	Entero largo	8	Utilizable con las columnas de tipo numérico, hora (listbox de tipo array únicamente) Tipo por defecto del resultado: Real

Note que los cálculos predefinidos tiene en cuenta todos los valores de la columna del list box, incluyendo los valores de las posibles líneas ocultas. Si desea restringir un cálculo a las líneas visibles, debe utilizar la constante lk footer custom y efectuar un cálculo personalizado.

Si el tipo de datos de la columna o de al menos una columna del list box (si objeto designa un list box) no es compatible con el calculo definido, el pie no se modifica y se genera el error 18. Si una columna contiene una fórmula (list box de tipo selección), se genera el error 10.

Nota: *las variables del área pie se definen automáticamente (cuando no se definen por programación) en función del tipo de cálculo definido en la Lista de propiedades (****Propiedades específicas de los pies de list box****). Si el tipo de la variable no corresponde al resultado esperado por el comando* ***LISTBOX SET FOOTER CALCULATION****, se genera un error. Por ejemplo, para una columna que muestra fechas, si el pie hace un cálculo 'Maximum', la variable pie será definida en fecha. Si ejecuta la instrucción* ***LISTBOX SET FOOTER CALCULATION*** *(pie;lk footer count), se genera un error por el tipo del resultado esperado (entero largo) difiere del tipo de la variable.*

LISTBOX SET FOOTERS HEIGHT

LISTBOX SET FOOTERS HEIGHT ({* ;} objeto ; altura {; unidad})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
altura	Entero largo	➡	Alto de la línea
unidad	Entero largo	➡	Unidad de valor de altura: 0 o si se omite = píxeles, 1 = líneas

Descripción

El comando ***LISTBOX SET FOOTERS HEIGHT*** *modifica por programación la altura de la línea de pie del list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena. Puede designar indiferentemente el list box o todo pie del list box.*

Pase en el parámetro altura la altura a definir. Por defecto, si omite el parámetro unidad, esta altura se expresa en píxeles. Para definir una unidad diferente, puede pasar una de las siguientes constantes (del tema ***Listbox****), en el parámetro unidad:*

Constante	Tipo	Valor	Comentario
lk lines	Entero largo	1	La altura designa un número de líneas. 4D calcula la altura de una línea en función de la fuente.
lk pixels	Entero largo	0	La altura es un número en píxeles (por defecto)

Nota*: para obtener más información sobre el cálculo de las alturas de líneas, consulte el manual de Diseño.*

LISTBOX SET GRID

LISTBOX SET GRID ({* ;} objeto ; horizontal ; vertical)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *) o Variable (si se omite *)
horizontal	Booleano	⇒	True = mostrar, False = ocultar
vertical	Booleano	⇒	True = mostrar, False = ocultar

Descripción

*El comando LISTBOX SET GRID permite mostrar u ocultar las líneas horizontales y/o verticales que componen la matriz del objeto list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección* ***Propiedades de los objetos****.*

*Pase en los parámetros horizontal y vertical los valores booleanos que indican si la líneas de la matriz deben mostrarse (****True****) u ocultarse (****False****). La matriz se muestra por defecto.*

LISTBOX SET GRID COLOR

LISTBOX SET GRID COLOR ({* ;} objeto ; color ; horizontal ; vertical)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
color	Entero largo	➡	Valor de color RGB
horizontal	Booleano	➡	Utilice el color par las líneas horizontales
vertical	Booleano	➡	Utilice el color par las líneas verticales

Descripción

El comando **LISTBOX SET GRID COLOR** le permite modificar el color de la rejilla del objeto list box designado por los parámetros objeto y *.

Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección .

Pase en el parámetro color un valor de color RGB. Para mayor información sobre los colores RGB, consulte la descripción del comando **SET RGB COLORS**.

Los parámetros horizontal y vertical le permiten especificar las líneas a las cuales quiere darle color:

- Si pasa **True** en horizontal, el color se aplicará a las líneas horizontales de la matriz. Si pasa **False**, el color de las líneas horizontales no cambiará.
- Si pasa **True** en vertical, el color se aplicará a las líneas verticales de la matriz. Si pasa **False**, el color de las líneas verticales no cambiará.

LISTBOX SET HEADERS HEIGHT

LISTBOX SET HEADERS HEIGHT ({* ;} objeto ; altura {; unidad})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
altura	Entero largo	➡	Alto de la línea
unidad	Entero largo	➡	Unidad de valor de altura: 0 o si se omite = píxeles, 1 = líneas

Descripción

El comando ***LISTBOX SET HEADERS HEIGHT*** *modifica por programación la altura de la línea de encabezado del list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

Puede designar indiferentemente el list box o todo encabezado del list box.

Pase en el parámetro altura la altura a definir. Por defecto, si omite el parámetro unidad, esta altura se expresa en píxeles. Para definir una unidad diferente, puede pasar una de las siguientes constantes (del tema ***Listbox****), en el parámetro unidad:*

Constante	Tipo	Valor	Comentario
lk lines	Entero largo	1	La altura designa un número de líneas. 4D calcula la altura de una línea en función de la fuente.
lk pixels	Entero largo	0	La altura es un número en píxeles (por defecto)

Los encabezados deben respetar la altura mínima establecida por el sistema. Esta altura es de 24 píxeles en Windows y 17 píxeles en Mac OS. Si pasa un valor más bajo en el parámetro de la altura, se aplica la altura mínima.

Nota*: para obtener más información sobre el cálculo de las alturas de líneas, consulte el manual de Diseño.*

LISTBOX SET HIERARCHY

LISTBOX SET HIERARCHY ({* ;} objeto ; jerarquico {; jerarquia})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
jerarquico	Booleano	➡	True = list box jerárquico False = list box no jerárquico
jerarquia	Array puntero	➡	Array de punteros

Descripción

*El comando **LISTBOX SET HIERARCHY** permite configurar el objeto list box designado por los parámetros objeto y * en modo jerárquico o no jerárquico.*

***Nota**: este comando sólo funciona con los list box basados en arrays. Cuando este comando se utiliza con un list box basado en las selecciones, no hace nada.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia variable en lugar de una cadena.*

El parámetro booleano jerarquico le permite especificar el modo del list box:

* *Si pasa True, el list box se muestra en modo jerárquico,*
* *Si se pasa False, el list box se muestra en modo no jerárquico (modo array estándar).*

*Cuando se pasa un list box en modo jerárquico, ciertas propiedades se restringen automáticamente. Para obtener más información, consulte la sección **Gestión de list box jerárquicos**.*

El parámetro jerarquia se utiliza para designar los arrays del list box a utilizar para la construcción de la jerarquía (ver ejemplo). Si muestra el list box en modo jerárquico y omite este parámetro:

- *Si el list box ya está en modo jerárquico, el comando no hace nada.*
- *Si el list box está en modo no jerárquico y nunca ha sido declarado jerárquico, el primer array se utiliza como la jerarquía por defecto.*
- *Si el list box está en modo no jerárquico, pero previamente ha sido declarado jerárquico, se restablece la última jerarquía.*

Ejemplo

Definición de los arrays aPais, aRegion y aCiudad como jerarquía de un list box:

```
ARRAY POINTER($ArrHierarch;3)
$ArrHierarch{1}:=->aPais  `Primer nivel de ruptura
$ArrHierarch{2}:=->aRegion  `Segundo nivel de ruptura
$ArrHierarch{3}:=->aCiudad  `Tercer nivel de ruptura
LISTBOX SET HIERARCHY(*;"mylistbox";True;$ArrHierarch)
```

LISTBOX SET LOCKED COLUMNS

LISTBOX SET LOCKED COLUMNS ({* ;} objeto ; numColumnas)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
numColumnas	Entero largo	➡	Número de columnas a bloquear

Descripción

El comando ***LISTBOX SET LOCKED COLUMNS*** *bloquea las primeras numColumnas columnas izquierdas del list box designado por los parámetros objeto y *.*

Las columnas bloqueadas se muestran en la parte izquierda del list box y no se desplazan con el resto de las columnas del list box. Para mayor información, consulte el Manual de Diseño.

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

En numColumnas, puede pasar cualquier valor entre 1 y el número total de columnas del list box -1. Para un list box con X columnas, si pasa un valor > X-1 en numColumnas, se reducirá automáticamente al valor X-1.

Para eliminar el bloqueo de columnas, pase 0 o un valor negativo en numColumnas.

LISTBOX SET PROPERTY

LISTBOX SET PROPERTY ({* ;} objeto ; propiedad ; valor)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable.
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (si se omite *)
propiedad	Entero largo	➡	Propiedad de list box o de columna
valor	Entero largo, Cadena	➡	Valor de la propiedad

Descripción

El comando ***LISTBOX SET PROPERTY*** *define el valor de la propiedad de la columna list box o list box especificada utilizando los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia variable en lugar de una cadena.*

Nota: *si el list box o la columna list box especificada utilizando los parámetros objeto y * no existe, el comando no hace nada y no se dispara ningún error.*

*En los parámetros propiedad y valor, se indica, respectivamente la propiedad a definir utilizando su nuevo valor. Puede utilizar las siguientes constantes del tema:"**Listbox**":*

Constante	Tipo	Valor	Comentario
lk allow wordwrap	*Entero largo*	*14*	*Propiedad **Wordwrap*** *Aplica a: Columna** *Valores posibles:* • *lk no (0)* • *lk yes (1)*
lk auto row height	*Entero largo*	*31*	*Propiedad **Altura de fila automática**.* *Aplica a: List box o columna* *Valores posibles:* • *lk yes* • *lk no*
lk background color expression	*Cadena*	*22*	***4D View Pro únicamente:** esta funcionalidad requiere una licencia 4D View Pro. Para más información, consulte **4D View Pro**.* ***Propiedad Background Color Expression** para list box de tipo selección* *Aplica a: List box o columna*
lk column max width	*Entero largo*	*26*	*Propiedad **Maximum Width*** *Aplica a: Columna**
lk column min width	*Entero largo*	*25*	
lk column resizable	*Entero largo*	*15*	*Propiedad **Resizable*** *Aplica a: Columna** *Valores posibles:* • *lk no (0)* • *lk yes (1)*
lk detail form name	*Cadena*	*19*	*Propiedad **Detail Form Name** para la selección de tipo list box* *Aplica a: List box*
lk display type	*Entero largo*	*21*	*Propiedad **Display Type** para columnas numéricas* *Aplica a: Columna** *Valores posibles:* • *lk numeric format (0): muestra valores en formato numérico* • *lk three states checkbox (1): muestra valores como casillas de selección de tres estados*
lk double click on row	*Entero largo*	*18*	
lk extra rows	*Entero largo*	*13*	*Propiedad **Hide extra blank rows*** *Aplica a: List box* *Valores posibles:* • *lk display (0)* • *lk hide (1)*
lk font color expression	*Cadena*	*23*	*Propiedad **Font Color Expression** para list box de tipo selección* *Aplica a: List box o columna*
lk font style expression	*Cadena*	*24*	
lk hide selection highlight	*Entero largo*	*16*	
lk highlight set	*Cadena*	*27*	
lk multi style	*Entero largo*	*30*	*Propiedad **Multiestilo*** *Aplica a: Columna** *Valores posibles:* • *lk no (0)[#/note]* • *lk yes (1) [#/note]*
lk named selection	*Cadena*	*28*	*Propiedad **Named Selection** para list box de tipo selección* *Aplica a: List box*
lk resizing mode	*Entero largo*	*11*	
lk row height unit	*Entero largo*	*17*	
lk selection mode	*Entero largo*	*10*	*Propiedad **Selection Mode*** *Aplica a: List box* *Valores posibles:* • *lk none (0)* • *lk single (1)* • *lk multiple (2)*

Constante	Tipo	Valor	Comentario
lk single click edit	*Entero largo*	*29*	*Propiedad* ***Single-Click Edit*** *Aplica a: List box* *Posible valores:* • *lk no (0)* • *lk yes (1)*
lk sortable	*Entero largo*	*20*	***Propiedad*** *Sortable* *Aplica a: List box* *Valores posibles:* • *lk no (0)* • *lk yes (1)*
lk truncate	*Entero largo*	*12*	*Propiedad* ***Truncate with ellipsis*** *Aplica a: List box o columna* *Valores posibles:* • *lk without ellipsis (0)* • *lk with ellipsis (1)*

Estas propiedades sólo se pueden aplicar a columnas list box; Sin embargo, si pasa un list box como parámetro,* ***LISTBOX SET PROPERTY* *aplica la propiedad a cada columna del list box.*

Nota: *si pasa una propiedad que no existe, o que no está disponible para el list box o columna especificado, por ejemplo lk font style expression en el caso de un list box de tipo array, el comando no hace nada y no se dispara ningún error.*

Ejemplo 1

Usted quiere asegurarse de que todas las columnas del list box "MyListbox" sean redimensionables:

```
LISTBOX SET PROPERTY(*;"MyListbox";lk column resizable;lk yes) //Todas las columnas del list box "MyListbox" se definen como redimensionables
```

Ejemplo 2

Usted desea definir el ancho máximo de la columna "ProductNumber":

```
LISTBOX SET PROPERTY(*;"ProductNumber";lk column max width;200) //Esta columna tendrá un ancho máximo de 200
```

LISTBOX SET ROW COLOR

LISTBOX SET ROW COLOR ({* ;} objeto ; fila ; color {; tipoColor})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
fila	Entero largo	➡	Numero de fila
color	Entero largo	➡	Valor de color RGB
tipoColor	Entero largo	➡	color de fuente del Listbox (por defecto) o color de fondo del Listbox

Descripción

Nota: *Este comando sólo funciona con listboxes de tipo array.*

El comando ***LISTBOX SET ROW COLOR*** *define el color de una fila o una celda en el listbox de tipo array designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

Puede designar un listbox o una columna del listbox en el parámetro objeto:

- *Cuando el objeto designa un listbox, el comando se aplica a la fila.*
- *Cuando el objeto designa una columna del listbox, el comando se aplica a la celda localizada en la intersección de columna/fila.*

En fila, pase el número de la fila en la que desea aplicar el nuevo color.

Nota: *El comando no tiene en cuenta ningún estado oculto/visible de las filas del listbox.*

En color, pase un valor de color RGB. Para obtener más información acerca de los colores RGB, consulte la descripción del comando ***OBJECT SET RGB COLORS****. Si desea que la fila herede el conjunto de colores para el nivel más alto, usted puede pasar la constante lk inherited en el parámetro color.*

En el parámetro tipoColor, pase las constantes lk background color o lk font color para indicar si desea aplicar el color para el fondo o para el color de fuente de la fila. Si omite este parámetro, el color se aplica para el color de fuente.

Este comando modifica los valores que se encuentran en el array de colores que pueden haberse definido para la columna del listbox. Si estos arreglos no están ya definidos, el comando crea dinámicamente arrays que puede acceder mediante el comando ***LISTBOX Get array****.*

Si los valores de color en conflicto se definen usando las propiedades para los listboxes o columnas del listbox, se aplica un orden de prioridad. Para obtener más información, consulte el manual de Diseño.

Ejemplo

En un array de tipo list box, queremos establecer los colores de una fila y para una celda de esta fila:

```
// Definition of font color for cell (yellow)
LISTBOX SET ROW COLOR(*;"Col5";3;0x00FFFF00)

// Definición de fondo y color de fuente para la fila 3
// Fondo rojo, fuente azul

LISTBOX SET ROW COLOR(*;"ListBox";3;0x00FF0000;lk background color)
LISTBOX SET ROW COLOR(*;"List Box";3;0x000000FF)
```

text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

LISTBOX SET ROW FONT STYLE

LISTBOX SET ROW FONT STYLE ({* ;} objeto ; fila ; estilo)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Variable (si * se omite)
fila	Entero largo	➡	Número de fila
estilo	Entero largo	➡	Estilo de fuente

Descripción

Nota: este comando sólo funciona con los list boxes de tipo array.

El comando **LISTBOX SET ROW FONT STYLE** establece un estilo de fuente para una fila o una celda en el list box tipo array designado por los parámetros objeto y * .

Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.

Puede designar un list box o una columna de list box en el parámetro de objeto:

- Cuando el objeto designa un list box, el comando se aplica a la fila.
- Cuando el objeto designa una columna de list box, el comando se aplica a la celda ubicada en la intersección de la columna/fila.

En fila, pase el número de la fila en la que desea aplicar el nuevo estilo.

Nota: el comando no tiene en cuenta ningún estado oculto/visible de las filas del list box.

En estilo, se pasa un valor de estilo. Debe utilizar uno (o una combinación) de las constantes que se encuentran en el tema **Estilos de fuente**:

Constante	Tipo	Valor
Bold	Entero largo	1
Italic	Entero largo	2
Plain	Entero largo	0
Underline	Entero largo	4

Si un array de estilos de fuente se ha asociado con el list box o columna, sólo el elemento correspondiente a la fila se modifica. En otras palabras, la ejecución del comando tiene el mismo efecto, en este caso, como modificación de un elemento del array estilo de fuente.
Si no hay un array estilo de fuente asociado con el list box o columna, se creará de forma dinámica cuando se llama a este comando. Se puede acceder a ellos usando el comando **LISTBOX Get array**.

Si las propiedades de estilo en conflicto se establecen para la columna o el list box, se aplica un orden de prioridad. Para obtener más información, consulte el manual de Diseño.

Nota: dado que los estilos de array para las columnas tienen prioridad sobre los de los list boxes, al aplicar este comando a un list box, sólo tendrá efecto si no hay estilo de array asignado a las columnas.

Ejemplo

Dado un array de tipo list box con las siguientes características:

- un array de estilo de fuente asociado con el list box (ArrGlobalStyle)
- un array de estilo de fuente asociado con la columna 5 (ArrCol5Style)
- las otras columnas no tienen arrays de estilo.

```
LISTBOX SET ROW FONT STYLE(*;"Col5";3;Bold)
 // equivalente a ArrCol5Style{3}:=Bold
```

text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	**text**	text
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

```
LISTBOX SET ROW FONT STYLE(*;"List Box";3;Italic+Underline)
 // equivalente a ArrGlobalStyle{3}:=Italic+Underline
```

text	text	text	text	text	text
text	text	text	text	text	text
text	*text*	*text*	*text*	**text**	*text*
text	text	text	text	text	text
text	text	text	text	text	text
text	text	text	text	text	text

Después de la segunda declaración, todas las celdas de la tercera fila cambian de fila a subrayadas en cursiva, a excepción de la celda de la quinta columna que queda sólo en negrilla (los arrays de estilo de columna tienen prioridad sobre los arrays de list box).

LISTBOX SET ROW HEIGHT

LISTBOX SET ROW HEIGHT ({* ;} objeto ; linea ; altura)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre objeto (si * se especifica) o Variable (si * se omite)
linea	Entero largo	➡	Línea de list box cuya altura desea definir
altura	Entero largo	➡	Altura de línea de list box

4D View Pro

*Este comando requiere una licencia 4D View Pro. Si esta licencia no está disponible, se muestra un error en el list box cuando el formulario se ejecuta. Para más información, consulte la sección **4D View Pro**.*

Descripción

*El comando **LISTBOX SET ROW HEIGHT** le permite modificar la altura de la fila especificada en el objeto list box designado utilizando los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia variable en lugar de una cadena. Para más información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.*

Si la fila especificada no existe en el list box, el comando no hace nada.

*La unidad utilizada para la altura especificada corresponde a la definida globalmente para las filas del list box, ya sea en la lista de propiedades o por una llamada anterior al comando **LISTBOX SET ROWS HEIGHT**.*

***Nota**: para más información sobre el cálculo de la altura de las filas, consulte el Manual de Diseño.*

*Si no hay un array de altura de fila asociado al list box, este comando crea uno dinámico; de lo contrario, se utiliza el array existente especificado en la propiedad **Array de altura de fila**. El uso de este comando para establecer el alto de fila individual produce el mismo resultado visual que la asociación de un array de altura de fila; Sin embargo, llenar un array con valores de altura de la fila es mucho más rápido que llamar a este comando en un bucle para ajustar la altura de las filas de una en una para el list box.*

***Nota importante:** si el comando **LISTBOX SET ROWS HEIGHT** se llama posteriormente con una unidad diferente a la que se definió previamente, este valor predeterminado reemplazará e reinicializará el array de altura de las filas existentes definidas utilizando **LISTBOX SET ROW HEIGHT** (ver el ejemplo).*

Ejemplo 1

Usted desea cambiar la altura de unas pocas líneas en el siguiente list box:

RowNum	Countries	Population
1	Luxembourg	502 202
2	Latvia	1 973 700
3	Kuwait	4 044 500
4	Croatia	4 284 889
5	Denmark	5 699 220
6	Nicaragua	6 071 045
7	Serbia	7 306 677
8	Honduras	8 249 574
9	Austria	8 572 895
10	Hungary	10 005 000
11	Czech Republic	10 674 947

Si ejecuta este código:

```
//la unidad actual es píxeles
LISTBOX SET ROW HEIGHT(*;"listboxname";3;40) //Kuwait
LISTBOX SET ROW HEIGHT(*;"listboxname";7;14) //Serbia
```

... obtiene el siguiente resultado:

RowNum	Countries	Population
1	Luxembourg	502 202
2	Latvia	1 973 700
3	Kuwait	4 044 500
4	Croatia	4 284 889
5	Denmark	5 699 220
6	Nicaragua	6 071 045
7	Serbia	7 306 677
8	Honduras	8 249 574
9	Austria	8 572 895
10	Hungary	10 005 000
11	Czech Republic	10 674 947

Ejemplo 2

Ha definido una altura de fila predeterminada y luego define varios valores de altura de fila individuales utilizando el comando ***LISTBOX SET ROW HEIGHT****:*

```
LISTBOX SET ROWS HEIGHT(*;"listboxname";25;lk pixels)  // altura global definida en píxeles
LISTBOX SET ROW HEIGHT(*;"listboxname";1;30)  // línea 1: 30 píxeles
LISTBOX SET ROW HEIGHT(*;"listboxname";5;40)  // línea 5: 40 píxeles
LISTBOX SET ROW HEIGHT(*;"listboxname";11;50)  //  línea 11: 50 píxeles
```

Más tarde, si se ejecuta el código siguiente...

```
LISTBOX SET ROWS HEIGHT(*;"listboxname";18;lk pixels)
```

…Entonces la altura global de fila se establece en 18 píxeles; Sin embargo, puesto que la unidad no ha cambiado, las filas 1, 5 y 11 mantendrán sus valores de altura específicos, es decir, 30, 40 y 50 píxeles como se definió anteriormente por el comando ***LISTBOX SET ROW HEIGHT****.*

Por otro lado, si el código siguiente se ejecuta posteriormente...

```
LISTBOX SET ROWS HEIGHT(*;"listboxname";2;lk lines)
```

... Entonces las filas 1, 5 y 11 se ponen a la altura global de fila predeterminada establecida por ***LISTBOX SET ROWS HEIGHT*** *(es decir, 2 líneas) ya que la unidad ha cambiado de píxeles a líneas. Puesto que no hay conversión automática aplicada, el cambio de unidades siempre resulta en altos de fila reinicializados en el nuevo valor por defecto definido.*

LISTBOX SET ROWS HEIGHT

LISTBOX SET ROWS HEIGHT ({* ;} objeto ; altura {; unidad})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
altura	Entero largo	➡	Altura de la fila (en píxeles)
unidad	Entero largo	➡	Unidad de valor de altura: 0 o se omite = píxeles, 1 = líneas

Descripción

El comando ***LISTBOX SET ROWS HEIGHT*** *le permite modificar por programación la altura de las filas del objeto list box designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección* ***Propiedades de los objetos****.*

Por defecto, si omite el parámetro unidad, la altura se expresa en píxeles. Para modificar la unidad, en el parámetro unidad pase una de las siguientes constantes, ubicadas en el tema ***Listbox****:*

Constante	Tipo	Valor	Comentario
lk lines	Entero largo	1	La altura designa un número de líneas. 4D calcula la altura de una línea en función de la fuente.
lk pixels	Entero largo	0	La altura es un número en píxeles (por defecto)

Nota: *para mayor información sobre el cálculo de las alturas de las líneas, consulte el manual de Diseño.*

LISTBOX SET STATIC COLUMNS

LISTBOX SET STATIC COLUMNS ({* ;} objeto ; numColumnas)

Parámetro	*Tipo*		*Descripción*
*	*Operador*	➡	*Si se especifica, objeto es un nombre de objeto (cadena)* *Si se omite, objeto es una variable*
objeto	*Objeto de formulario*	➡	*Nombre de objeto (si se especifica *) o* *Variable (si se omite *)*
numColumnas	*Entero largo*	➡	*Número de columnas a convertir estáticas*

Descripción

*El comando **LISTBOX SET STATIC COLUMNS** define las primeras numColumnas columnas (empezando por la izquierda) en el list box designado por los parámetros objeto y *.*

Las columnas estáticas no pueden moverse en el list box.

***Nota:** las columnas estáticas y las columnas bloqueadas son dos funcionalidades independientes. Para mayor información, consulte el manual de Diseño.*

LISTBOX SET TABLE SOURCE

LISTBOX SET TABLE SOURCE ({* ;} objeto ; numTabla | tempo {; nomSel})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite*)
numTabla | tempo	Entero largo, Cadena	➡	Número de la tabla de la cual utilizarla selección actual o nombre de la selección temporal a utilizar
nomSel	Cadena	➡	Nombre del conjunto seleccionado

Descripción

*El comando **LISTBOX SET TABLE SOURCE** permite modificar la fuente de datos mostrada en el listbox diseñado por los parámetros * y objeto*

***Nota:** este comando sólo puede utilizarse cuando la propiedad "Fuente de datos" del list box es **Selección actual** o **Selección temporal** (para mayor información, consulte la sección **Gestión programada de los objetos de tipo List box**). El comando no hace nada si lo utiliza con un list box asociado a un array, colecciones o selecciones de entidades.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, no pase una cadena sino una referencia de variable. Para mayor infomación sobre nombres de objetos, por favor consulte la sección indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.*

Si pasa un número de tabla como parámetro numTabla, el list box se llenará con los datos de los registros en la selección actual de la tabla.

Si pasa un nombre de selección temporal como parámetro tempo, el list box se llenará con los datos de los registros que pertenecen a la selección temporal.

El parámetro opcional nomSubrayado permite asociar un conjunto resaltado al list box. El conjunto resaltado se utiliza para la gestión de resaltar los registros por el usuario en el list box.

Si el list box ya contiene las columnas, sus contenidos se actualizarán después de la ejecución del comando.

***Nota**: por razones de optimización, este comando se procesa de manera asincrónica, es decir el cambio de fuente del listbox se realiza sólo cuando se completa la ejecución del método del cual se llama el comando.*

LISTBOX SORT COLUMNS

LISTBOX SORT COLUMNS ({* ;} objeto ; numColumna ; orden {; numColumna2 ; orden2 ; ... ; numColumnaN ; ordenN})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
numColumna	Entero largo	➡	Número(s) de columna(s) a ordenar
orden	Operador	➡	">" para una ordenación ascendente o "<" para una ordenación descendente

Descripción

*El comando **LISTBOX SORT COLUMNS** ordena las filas del list box designado por los parámetros objeto y * en función de los valores de una o varias columnas.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección indica que el parámetro objeto es un nombre de objeto (cadena). Si omite este parámetro, indica que el parámetro objeto es una variable. En ese caso, no pasa una cadena, sino una referencia de variable. Para mayor información sobre nombres de objetos, consulte la sección .*

En numCol, pase el número de columna cuyos valores quiere utilizar como criterio de ordenación. Puede utilizar todo tipo de datos, a excepción de imágenes y punteros.

En orden, pase el símbolo > o < para indicar el criterio de ordenación. Si orden contiene el símbolo "mayor que"(>), el criterio de ordenación es ascendente. Si orden contiene el símbolo "menor que" (<), el criterio de ordenación es descendente.

Puede definir ordenaciones multiniveles: para hacerlo, pase tantos pares (numCol;orden) como sea necesario. El nivel de ordenación se define por la posición del parámetro en la llamada.

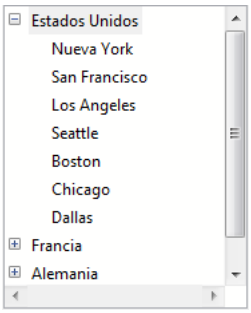
Conforme con el principio de funcionamiento del list box, las columnas están sincronizadas, lo que significa que la ordenación de una columna repercute automáticamente en todas las otras columnas del objeto.

Listas jerárquicas

- *Gestión de listas jerárquicas*
- *APPEND TO LIST*
- *CLEAR LIST*
- *Copy list*
- *Count list items*
- *DELETE FROM LIST*
- *Find in list*
- *GET LIST ITEM*
- *Get list item font*
- *GET LIST ITEM ICON*
- *GET LIST ITEM PARAMETER*
- *GET LIST ITEM PARAMETER ARRAYS*
- *GET LIST ITEM PROPERTIES*
- *GET LIST PROPERTIES*
- *INSERT IN LIST*
- *Is a list*
- *List item parent*
- *List item position*
- *LIST OF CHOICE LISTS*
- *Load list*
- *New list*
- *SAVE LIST*
- *SELECT LIST ITEMS BY POSITION*
- *SELECT LIST ITEMS BY REFERENCE*
- *Selected list items*
- *SET LIST ITEM*
- *SET LIST ITEM FONT*
- *SET LIST ITEM ICON*
- *SET LIST ITEM PARAMETER*
- *SET LIST ITEM PROPERTIES*
- *SET LIST PROPERTIES*
- *SORT LIST*
- *_o_REDRAW LIST*

Gestión de listas jerárquicas

Las listas jerárquicas son objetos de formulario que permiten mostrar los datos como listas con uno o más niveles que es posible expandir o contraer.



*En los formularios, las listas jerárquicas pueden servir para visualizar o introducir datos. Cada elemento de la lista puede contener hasta 2 millardos de caracteres (tamaño máximo de un campo de texto) y estar asociado con un icono. Las listas jerárquicas por lo general soportan clic, doble clic y navegación con el teclado como también arrastrar y soltar. Es posible efectuar una búsqueda en el contenido de una lista (comando **Find in list**).*

Creación y modificación

*Las listas jerárquicas pueden ser creadas completamente por programación (vía los comandos **New list** o **Copy list**) o utilizando listas definidas en el editor de listas en el entorno Diseño (comando **Load list**).*

*Los contenidos y la apariencia de las listas jerárquicas son administrados por programación utilizando los comandos del tema "Listas jerárquicas". Ciertas características de apariencia específicas también pueden ser definidas utilizando comandos genéricos del tema **Propiedades de los objetos** (ver a continuación).*
*Puede asociar dinámicamente las referencias de listas jerárquicas a las las listas de selección de objetos de formularios (fuentes, valores obligatorio y valores excluidos) utilizando los comandos **OBJECT SET LIST BY REFERENCE** o **OBJECT SET LIST BY NAME**. También puede asociar las listas de selección definidas en el editor de listas a los objetos de formularios vía la Lista de propiedades.*

ListRef y nombre de objeto

*Una lista jerárquica es a la vez un **objeto de lenguaje** existente en memoria y un **objeto de formulario**.*

*El **objeto de lenguaje** se referencia de manera única por un identificador interno, de tipo Entero largo, diseñado por **RefList** en este manual. Este identificador es devuelto por los comandos que pueden ser utilizados para crear listas: **New list**, **Copy list**, **Load list**, **BLOB to list**. Existe una sola instancia en memoria del objeto de lenguaje y toda modificación que se lleve a cabo en este objeto inmediatamente repercute en todos los lugares donde se utiliza.*

*El **objeto de formulario** no es necesariamente único: puede haber varias representaciones de una misma lista jerárquica en un mismo formulario o en formularios diferentes. Como para los otros objetos de formulario, usted especifica el objeto en el lenguaje utilizando la sintaxis (*;"NomLista", etc.).*

*Usted conecta el "objeto de lenguaje" lista jerárquica con el "objeto de formulario" por intermedio de la variable que contiene el valor del identificador único **RefLista**. Por ejemplo, si escribe:*

```
milista:=New list
```

*... es suficiente asociar el nombre de la variable milista al objeto de formulario Lista jerárquica en la lista de propiedades con el fin de que administre el objeto de lenguaje cuyo **RefLista** se guarda en milista*

Cada representación de la lista tiene características propias como también características en común con las otras representaciones. Las siguientes características son específicas para cada representación de lista:

- *La selección,*
- *El estado desplegado/contraído de sus elementos,*
- *La posición del cursor de desplazamiento,*

Las otras características (fuente, estilo de fuente, estilo, filtro de entrada, color, contenido de la lista, iconos, etc.) son comunes a todas las representaciones y no pueden ser modificadas por separado.

*Por lo tanto, cuando utilice comandos basados en la configuración expandida/contraída o el elemento actual, por ejemplo **Count list items** (cuando el parámetro * final no se pasa), es importante poder especificar sin ambigüedad la representación a utilizar.*

*Debe utilizar el identificador de tipo **RefLista** con los comandos del lenguaje cuando quiera especificar la lista jerárquica residente en memoria.*

Si quiere especificar la representación a nivel de formulario de un objeto Lista jerárquica, debe utilizar el nombre del objeto (tipo cadena) en el comando, vía la sintaxis (;"NomLista", etc.). Esta sintaxis es idéntica a la utilizada en los comandos del tema "Propiedades de los objetos". Es aceptada por la mayoría de los comandos del tema "Listas jerárquicas" que actúan sobre las propiedades de las listas (por favor consulte la descripción de los comandos de este tema).*

***Advertencia**, en el caso de los comandos que definen propiedades, la sintaxis basada en el nombre del objeto no significa que sólo el objeto de formulario especificado será modificado por el comando, sino que la acción del comando estará basada en el*

estado de este objeto. Las características comunes de las listas jerárquicas siempre son modificadas en todas las representaciones.

*Por ejemplo, si pasa la instrucción **SET LIST ITEM FONT(*;"milista1";*;lafuente)**, está indicando que quiere modificar la fuente de la lista jerárquica asociada al objeto de formulario milista1. El comando tendrá en cuenta el elemento actual del objeto milista1 para definir el elemento a modificar, pero esta modificación será llevada a cabo en todas las representaciones de la lista en todos los procesos.*

Soporte de @

Como para los otros comandos de gestión de propiedades de objetos, es posible utilizar el carácter "@" en el parámetro ListName. En principio, esta sintaxis se utiliza para designar un conjunto de objetos en el formulario. Sin embargo, en el contexto de los comandos de listas jerárquicas, este principio no aplica en todos los casos. Esta sintaxis tendrá dos efectos diferentes dependiendo del tipo de comando:

- *Para los comandos que definen las propiedades, esta sintaxis designa todos los objetos cuyo nombre corresponde (funcionamiento estándar). Por ejemplo, el parámetro "LH@" designa todos los objetos de tipo lista jerárquica cuyo nombre comienza con "LH." Estos comandos son:*
  ***DELETE FROM LIST*,**
  INSERT IN LIST
  SELECT LIST ITEMS BY POSITION
  SET LIST ITEM
  SET LIST ITEM FONT
  SET LIST ITEM ICON
  SET LIST ITEM PARAMETER
  SET LIST ITEM PROPERTIES
- *Para los comandos que recuperan propiedades, esta sintaxis designa el primer objeto cuyo nombre corresponde. Estos comandos son:*
  Count list items
  Find in list
  GET LIST ITEM
  Get list item font
  GET LIST ITEM ICON
  GET LIST ITEM PARAMETER
  GET LIST ITEM PROPERTIES
  List item parent
  List item position
  Selected list items

Comandos genéricos utilizables con listas jerárquicas

*Es posible modificar la apariencia de una lista jerárquica en un formulario utilizando varios comandos 4D genéricos. Debe pasar a estos comandos bien sea el nombre del objeto de la lista jerárquica (utilizando el parámetro *) o su nombre de variable (sintaxis estándar).*

***Nota:** en el caso de las listas jerárquicas, la variable del formulario contiene el valor de **RefLista**. Si ejecuta un comando que modifica un atributo pasándole la variable asociada a la lista jerárquica, no será posible definir la lista objetivo en el caso de las representaciones múltiples. Por lo tanto, sólo el nombre del objeto permite diferenciar individualmente cada representación.*

Esta es una lista de comandos que pueden utilizarse con las listas jerárquicas.

OBJECT SET FONT
OBJECT SET FONT STYLE
OBJECT SET FONT SIZE
OBJECT SET COLOR
OBJECT SET FILTER
OBJECT SET ENTERABLE
OBJECT SET SCROLLBAR VISIBLE
OBJECT SET SCROLL POSITION
OBJECT SET RGB COLORS

***Recuerde:** excepto por el comando **OBJECT SET SCROLL POSITION**, estos comandos modifican todas las representaciones de una misma lista, incluso si sólo especifica una lista vía su nombre de objeto.*

Prioridad de los comandos de propiedad

Algunas propiedades de las listas jerárquicas (por ejemplo, el atributo editable o color) pueden definirse de tres maneras: vía la Lista de propiedades en el entorno Diseño, vía un comando del tema "Propiedades de los objetos" o vía un comando del tema "Listas jerárquicas".

Cuando las tres maneras se utilizan para definir propiedades de una lista, se aplica el siguiente orden de prioridad:

1. Comandos del tema "Listas jerárquicas"
2. Comandos genéricos de propiedad del objeto
3. Parámetros de la Lista de propiedades

*Este principio se aplica sin importar el orden en el cual se llaman los comandos. Si se modifica una propiedad del elemento individualmente vía un comando de lista jerárquica, el comando de propiedad del objeto equivalente no tendrá efecto en este elemento incluso si se llama posteriormente. Por ejemplo, si modifica el color de un elemento vía el comando **SET LIST ITEM PROPERTIES**, el comando **OBJECT SET COLOR** no tendrá efecto sobre este elemento.*

Gestión de elementos por posición o por referencia

Generalmente puede trabajar de dos formas con los contenidos de las listas jerárquicas: por posición o por referencia.

- *Cuando trabaja por posición, 4D se basa en la posición relativa de los elementos en la lista mostrada en la pantalla para identificarlos. El resultado será diferente si los elementos jerárquicos están desplegados o contraídos. Note que en el caso de representaciones múltiples, cada objeto de formulario, tiene su propia configuración de elementos contraídos/desplegados.*
- *Cuando trabaja por referencia, 4D se basa en el número único refElemento de los elementos de la lista. Entonces cada elemento puede ser especificado individualmente, sin importar su posición o visualización en la lista jerárquica.*

Uso de los números de referencia de los elementos (refElemento)

Cada elemento de una lista jerárquica tiene un número de referencia (refElemento) de tipo Entero largo. Este valor está destinado únicamente a su propio uso: 4D simplemente lo mantiene.

***Advertencia:** como número de referencia puede utilizar todo valor de tipo Entero largo, excepto 0. De hecho, para la mayoría de los comandos de este tema, el valor 0 se utiliza para especificar el último elemento añadido a la lista.*

Estos son algunos consejos para utilizar números de referencia:

1. No necesita identificar cada elemento con un número único (nivel principiante).

- *Primer ejemplo: usted construye por programación un sistema de pestañas, por ejemplo, una libreta de direcciones. Como el sistema devuelve el número de la pestaña seleccionada, probablemente necesite más información. En este caso, no se preocupe por los números de referencia de los elementos: pase cualquier valor (excepto 0) en el parámetro refElemento. Note que para un sistema de libreta de direcciones, puede predefinir una lista A, B, ..., Z en el entorno Diseño. Igualmente puede crearla por programación para eliminar las letras para las cuales no hay registros.*

- *Segundo ejemplo: trabajando con una base, usted construye progresivamente una lista de palabras claves. Puede guardar esta lista al final de cada sesión utilizando los comandos **SAVE LIST** o **LIST TO BLOB** y cargarla nuevamente al inicio de cada nueva sesión utilizando los comandos **Load list** o **BLOB to list**. Puede mostrar esta lista en una paleta flotante; cuando el usuario hace clic en una palabra clave de la lista, el elemento elegido se inserta en el área editable seleccionada del proceso del primer plano. También puede utilizar arrastrar y soltar. En todos los casos, lo importante es que usted sólo procesa el elemento seleccionado (por clic o arrastrar y soltar), porque los comandos **Selected list items** (en el caso de un clic) y **DRAG AND DROP PROPERTIES** devuelven la posición del elemento que usted debe procesar. Cuando utilice este valor de posición, obtiene el título del elemento gracias al comando **GET LIST ITEM**. Acá tampoco, necesita identificar cada elemento individualmente; puede pasar todo valor (excepto 0) en el parámetro refElemento.*

2. Debe identificar parcialmente los elementos de la lista (nivel intermedio).

*Utilice el número de referencia del elemento para guardar la información necesaria cuando trabaje con el elemento; este punto se detalla en el ejemplo del comando **APPEND TO LIST**. En este ejemplo, utilizamos los números de referencia del elemento para guardar los números de registros. Sin embargo, debemos poder establecer una distinción entre los elementos que corresponden a los registros [Departamentos] y los que corresponden a los registros [Empleados].*

3. Debe identificar individualmente los elementos de la lista (nivel avanzado).

*Usted programa una gestión elaborada de listas jerárquicas en las cual debe poder identificar individualmente cada elemento en todos los niveles de la lista. Una manera sencilla de implementar este funcionamiento es mantener un contador personal. Supongamos que crea una lista hlList utilizando el comando **New list**. En este momento, usted inicializa un contador vhlCounter en 1. Cada vez que usted llama **APPEND TO LIST** o **INSERT IN LIST**, se incrementa este contador (vhlCounter:=vhlCounter+1), y usted pasa el número del contador como número de referencia del elemento. El truco consiste en no disminuir el contador cuando borre elementos — el contador solo puede crecer. De esta forma, usted garantiza la unicidad de los números de referencia de los elementos. Como estos números son de tipo Entero largo, puede añadir o insertar más de dos millardos de elementos en una lista que ha sido reinicializada... (Sin embargo si está trabajando con grandes cantidades de elementos, esto generalmente significa que debe utilizar una tabla en lugar de una lista.)*

***Nota**: si utiliza **Operadores de bits**, igualmente puede utilizar los números de referencia de los elementos para almacenar la información que puede ser alojada en un Entero largo, es decir 2 Enteros, valores de 4 bytes o 32 Booleanos.*

¿Cuándo necesita números de referencia únicos?

*En la mayoría de los casos, cuando utiliza listas jerárquicas en interfaces de usuario que sólo tratan con el elemento seleccionado (el cual recibió un clic o fue arrastrado), no necesitará utilizar los números de referencia de los elementos. La utilización de los comandos **Selected list items** y **GET LIST ITEM** le proporcionan toda la información necesaria para la gestión del elemento seleccionado actualmente. Además, los comandos tales como **INSERT IN LIST** y **DELETE FROM LIST** le permiten manipular la lista de manera "relativa" al elemento seleccionado.*

Básicamente, necesita tratar con los números de referencia de los elementos cuando quiera acceder directamente por programación a cualquier elemento de la lista y no necesariamente al elemento actualmente seleccionado.

APPEND TO LIST

APPEND TO LIST (lista ; textoElem ; refElem {; sublista ; desplegada})

Parámetro	Tipo		Descripción
lista	ListRef	➡	Número de referencia de lista
textoElem	Cadena	➡	Texto del nuevo elemento de lista (max. 255 caracteres)
refElem	Entero largo	➡	Número de referencia único del nuevo elemento
sublista	ListRef	➡	Sublista opcional para añadir al nuevo elemento
desplegada	Booleano	➡	Indica si las sublistas opcionales serán desplegadas o contraídas

Descripción

El comando APPEND TO LIST añade un nuevo elemento a la lista jerárquica cuyo número de referencia se pasa en lista.

El texto del elemento se pasa en textoElem. Puede pasar una expresión de tipo Alfa o Texto de máximo 2 000 000 caracteres. A partir de 4D v16 R4, si el elemento está asociado con una acción estándar, puede pasar la constante ak standard action title en textoElem para usar automáticamente el nombre de la acción localizado. Para más información, consulte la sección ***Acciones estándar****.*

El número de referencia único del elemento (del tipo Entero largo) se pasa en refElem. Aunque clasificamos este número de referencia como único, en realidad puede pasar el valor que quiera. Consulte la sección ***Gestión de listas jerárquicas*** *para mayor información sobre el parámetro refElem.*

Igualmente si quiere que un elemento tenga elementos hijos, pase un número de referencia de lista válido en el parámetro sublista. En este caso, también debe pasar el parámetro expandido. Pase ***True*** *o* ***False*** *en este parámetro de manera que la sublista se muestre desplegada o contraída respectivamente.*

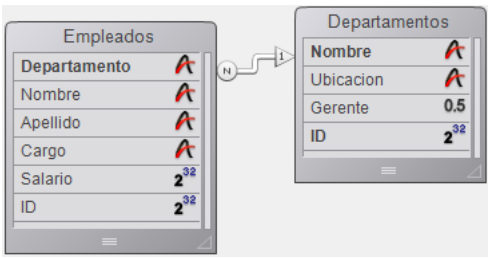
La referencia de la lista que pasa en sublista debe hacer referencia a una lista existente. La lista existente podría tener un solo nivel o tener sublistas. Si no quiere añadir una lista hija al nuevo elemento, omita el parámetro o pase 0. Aunque ambos son opcionales, los parámetros sublista y desplegada deben pasarse de forma conjunta.

Consejos:

- *Para insertar un nuevo elemento en una lista, utilice* ***INSERT IN LIST****. Para cambiar el texto de un elemento existente o modificar su lista hija al igual que su estado desplegado, utilice* ***SET LIST ITEM****.*
- *Para cambiar la apariencia del nuevo elemento añadido utilice* ***SET LIST ITEM PROPERTIES****.*

Ejemplo

Esta es una vista parcial de la estructura de una base:



Las tablas [Departamentos] y [Empleados] contienen los siguientes registros:

Nombre :	Ubicación :	Gerente :
Biologia marina	Primer piso	Roberto Pérez
Contabilidad	Segundo piso	Gerardo Díaz
Ventas	Segundo piso	Eliana López

Departamento :	Nombre :	Apellido :
Biologia marina	Dario	Villa
Contabilidad	Gerardo	Pérez
Ventas	Gustavo	Casas
Contabilidad	Gilberto	Cardenas
Biologia marina	Alvaro	Suarez
Ventas	Federico	Mendieta
Biologia marina	Pedro	Martinez

Usted quiere mostrar una lista jerárquica, llamada hlList, que muestre los departamentos y para cada departamento, un lista hija que muestre los empleados que trabajan en ese departamento. El método de objeto de hlList es:

```
// Método de objeto de lista jerárquica hlList

Case of

   :(Form event=On Load)
      C_LONGINT(hlList;$hSubList;$vlDepartament;$vlEmpleado;$v1DepartmentID)
// Crear una nueva lista jerárquica vacía
      hlList:=New list
```

```
  // Seleccionar todos los registros de la tabla [Departamentos]
        ALL RECORDS([Departamentos])
  // Para cada departamento
        For($vlDepartament;1;Records in selection([Departamentos]))
  // Seleccionar los empleados de ese departamento
           RELATE MANY([Departamentos]Nombre)
  // ¿Cuántos hay?
           $vlNbEmpleados:=Records in selection([Empleados])
  // ¿Hay por lo menos un empleado en este departamento?
           If($vlNbEmpleados>0)
  // Crear una lista hija para el elemento Departamento
              $hSubList:=New list
  // Para cada Empleado
              For($vlEmpleado;1;Records in selection([Empleados]))
  // Añadir el elemento Empleado a la sublista
  // Note que el campo ID del registro [Empleados]
  // se pasa como número de referencia del elemento
                 APPEND TO LIST($hSubList;[Empleados]Apellido+", "+[Empleados]Nombre;[Empleados]ID)
  // Ir al siguiente registro [Empleados]
                 NEXT RECORD([Empleados])
              End for
           Else
  // No Empleados, no lista hija para el elemento Departamento
              $hSubList:=0
           End if
  // Añadir el elemento Departamento a la lista principal
  // Note que el número del registro [Departamentos]
  // se pasa como número de referencia del elemento. El bit #31
  // del número de referencia del elemento es forzado a uno de manera que podamos
  // distinguir entre los elementos Departamentos y Empleados. Ver nota sobre por qué
  // podemos utilizar este bit como información suplementaria sobre el elemento.
           APPEND TO LIST(hlList;[Departamentos]Nombre;[Departamentos]ID?+31;$hSublist;$hSubList #0)
  // Asignar el elemento Departamento en negrita para enfatizar la jerarquía de la lista
           SET LIST ITEM PROPERTIES(hlList;0;False;Bold;0)
  // Ir al siguiente Departamento
           NEXT RECORD([Departamentos])
        End for
  // Ordenar toda la lista en orden ascendente
        SORT LIST(hlList;>) // Mostrar la lista utilizando el estilo Windows
  // y forzar la altura de línea mínima a 14 Pts
        SET LIST PROPERTIES(hlList;Ala Windows;Windows node;14)

     :(Form event=On Unload)
  // La lista ya no es necesaria; ¡No olvide borrarla!
        CLEAR LIST(hlList;*)

     :(Form event=On Double Clicked)
  // Hay un doble clic
  // Obtener la posición del elemento seleccionado
        $vlItemPos:=Selected list items(hlList)
  // Verificar la posición
        If($vlItemPos #0)
  // Obtener la información del elemento de la lista
           GET LIST ITEM(hlList;$vlItemPos;$vlItemRef;$vsItemText;$vlItemSubList;$vbItemSubExpanded)
  // ¿Este elemento es elemento de un Departamento?
           If($vlItemRef ?? 31)
  // Si es así, es un doble clic en un elemento Departamento
              ALERT("Usted hizo doble clic en el elemento Departamento "+Char(34)+$vsItemText+Char(34)+".")
           Else
  // Si no, es un doble clic en un elemento Empleado
  // Utilizando el número de referencia del elemento padre encontrar el registro [Departamentos]
              $v1DepartmentID:=List item parent(hlList;$vlItemRef)?-31)
              QUERY([Departamentos];[Departmentos]ID=$vlDepartmentID)
  // Informar donde trabaja el Empleado y a quién le reporta
              ALERT("Usted hizo doble clic en el elemento Empleado "+Char(34)+$vsItemText+Char(34)+" que trabaja en el Departamento
"+Char(34)+
              [Departamentos]Nombre+Char(34)+" cuyo gerente es "+Char(34)+[Departamentos]Gerente+Char(34)+".")
           End if
        End if

 End case
```

```
// Nota: 4D puede almacenar hasta 1 000 millones de registros por tabla.
// En nuestro ejemplo, utilizamos el bit #31 del byte superior no utilizado para
// diferenciar los elementos de Empleados y Departamentos.
```

En este ejemplo, sólo hay una razón para establecer una diferencia entre los elementos [Departamentos] y [Empleados]:

1) Almacenamos números de registros en los números de referencia de los elementos; por lo tanto, probablemente terminaremos con elementos [Departamentos] cuyo número de referencia de elemento son los mismos que los de los elementos [Empleados].

*2) Utilizamos el comando **List parent item** para recuperar el padre del elemento seleccionado. Si hacemos clic en un elemento [Empleados] cuyo número de registro asociado es 10, y si existe también un elemento [Departamentos] que tiene el número 10, el elemento [Departamentos] será encontrado primero por **List parent item** cuando esta función analice la lista para ubicar el elemento con el número de referencia del elemento que pasamos. El comando devolverá el padre del elemento [Departamentos] y no el padre del elemento [Empleados].*

Por lo tanto, hemos hecho que los números de referencia de los elementos sean únicos, no porque queramos número únicos, si no por que necesitamos diferenciar los elementos de [Departamentos] y [Empleados].

Cuando el formulario se ejecuta, la lista se verá de esta forma:

***Nota:** este ejemplo es útil para propósitos de interfaz de usuario si trabaja con un número limitado de registros. Recuerde que las listas se conservan en memoria, no construya interfaces de usuario con listas jerárquicas que contengan millones de elementos.*

CLEAR LIST

CLEAR LIST (lista {; *})

Parámetro	Tipo		Descripción
lista	ListRef	➡	Número de referencia de la lista
*		➡	Si se especifica, si hay sublistas las borra de la memoria, Si se omite, las sublistas no son borradas

Descripción

*El comando **CLEAR LIST** borra de la memoria la lista jerárquica cuyo número de referencia se pasa en lista.*

*Generalmente debe pasar el parámetro opcional *, de manera que todas las sublistas, si las hay, asociadas a los elementos o subelementos de la lista también sean borradas.*

*No es necesario borrar una lista asociada a un objeto de formulario por medio de la ventana de la Lista de propiedades. 4D carga y borra la lista por usted. Por otra parte, cada vez que usted carga, copia, extrae de un BLOB, o crea una lista por programación, llama al comando **CLEAR LIST** cuando usted no necesita más la lista.*

Para borrar un sublista asociada a un elemento (en cualquier nivel) de otra lista mostrada actualmente en un formulario, proceda de la siguiente forma:

*1. Llame **GET LIST ITEM** con el elemento padre para obtener el número de referencia de la sublista.*
*2. Llame **SET LIST ITEM** con el elemento padre para separar la sublista del elemento de la lista antes de borrarlos.*
*3. Llame **CLEAR LIST** para borrar la sublista cuyo número de referencia obtuvo con **GET LIST ITEM**.*
*4. Llame **REDRAW LIST** para la lista mostrada en el formulario, para recalcular sus elementos y sublistas.*

Ejemplo 1

*Dentro de una rutina de limpieza que borra todos los objetos y datos que ya no necesita (por ejemplo, cuando se cierra una ventana o un formulario), usted podría terminar borrando una lista jerárquica que ya haya sido borrada, dependiendo de las acciones del usuario en el formulario. Utilice **Is a list** para borrar la lista sólo si es necesario:*

```
` Extraer de la rutina de limpieza
If(Is a list(hlList))
   CLEAR LIST(hlList;*)
End if
```

Ejemplo 2

*Ver el ejemplo del comando **Load list**.*

Ejemplo 3

*Ver el ejemplo del comando **BLOB to list**.*

Copy list

Copy list (lista) -> Resultado

Parámetro	Tipo		Descripción
lista	ListRef	→	Número de referencia de la lista a copiar
Resultado	ListRef	↩	Número de referencia de la nueva lista

Descripción

*El comando **Copy list** duplica la lista cuyo número de referencia se pasa en el parámetro lista y devuelve el número de referencia de la nueva lista.*

*Una vez hay terminado de utilizar la nueva lista, llame a **CLEAR LIST** para borrarla.*

Count list items

Count list items ({* ;} lista {; *}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
*	Operador	➡	Si se omite (por defecto): Devuelve los elementos visibles (desplegados) de la lista Si se especifica: Devuelve todos los elementos de la lista
Resultado	Entero largo	⮌	Número de elementos visibles de la lista (desplegados) (si se omite el segundo *) o número total de elementos de la lista (si se presenta el segundo *)

Descripción

*El comando **Count list items** devuelve el número actual de elementos visibles o el número total de elementos en la lista cuyo número de referencia o nombre de objeto se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, usted indica que el parámetro lista es una referencia de lista jerárquica (refLista). Si utiliza una sola representación de lista o trabaja con todos los elementos (pasa el segundo *), puede utilizar cualquiera de las dos sintaxis. Por el contrario, si usted utiliza varias representaciones de la misma lista y trabaja con los elementos visibles (el segundo * se omite), la sintaxis basada en el nombre del objeto es necesaria ya que cada representación puede tener su propia configuración desplegada/contraída.*

***Nota:** si utiliza el carácter @ en el nombre del objeto de la lista y el formulario contiene varias listas que coinciden con este nombre, el comando **Count list items** se aplicará al primer objeto cuyo nombre corresponda.*

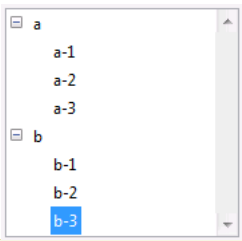
*Utilice el segundo parámetro * para determinar que tipo de información se devolverá. Cuando se pasa este parámetro, el comando devuelve el número total de elementos presentes en la lista, sin importar si la lista está desplegada o contraída.*

Cuando se omite este parámetro, el comando devuelve el número de elementos visibles, dependiendo del estado expandido/contraído de la lista y sus sublistas.

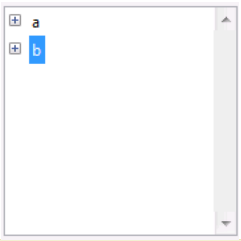
Aplique este comando a una lista mostrada en un formulario.

Ejemplos

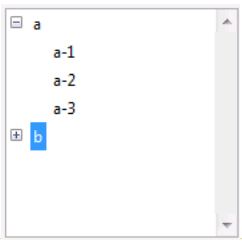
Acá tenemos una lista llamada hList mostrada en el entorno Aplicación:




```
$vlNbItems:=Count list items(hList) ` en este punto $vlNbItems vale 8
$vlNbTItems:=Count list items(hList;*) `$vlNbTItems también vale 8
```




```
$vlNbItems:=Count list items(hList) ` en este punto $vlNbItems vale 2
$vlNbTItems:=Count list items(hList;*) `$vlNbTItems continúa valiendo 8
```




```
$vlNbItems:=Count list items(hList) ` en este punto $vlNbItems vale 5
$vlNbTItems:=Count list items(hList;*) `$vlNbTItems continúa valiendo 8
```

DELETE FROM LIST

DELETE FROM LIST ({* ;} lista ; refElem | * {; *})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
refElem | *	Entero largo, Operador	➡	Número de referencia del elemento, o 0 para el último elemento añadido a la lista o * para el elemento de la lista actualmente seleccionada
*		➡	Si se especifica, borra las sublistas (si la hay) de la memoria Si se omite, las sublistas (si las hay) no son borradas

Descripción

*El comando **DELETE FROM LIST** borra el elemento designado por el parámetro refElem de la lista cuyo número de referencia se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, usted indica que el parámetro lista es una referencia de lista jerárquica (refLista). Si utiliza una sola representación de lista o trabaja con todos los elementos (pasa el segundo *), puede utilizar cualquiera de las dos sintaxis. Por el contrario, si utiliza varias representaciones de la misma lista y trabaja con los elementos visibles (el segundo * se omite), la sintaxis basada en el nombre del objeto es necesaria ya que cada representación puede tener su propia configuración desplegada/contraída.*

*Si pasa * en refElem, borra el elemento actualmente seleccionado en la lista. Igualmente puede pasar 0 en este parámetro para borrar el último elemento añadido a la lista.*

De lo contrario, especifique el número de referencia del elemento que quiere borrar. Si el número no corresponde a ningún elemento de la lista, el comando no hace nada.

*Si trabaja con los números de referencia de los elementos, construya una lista en la cual los elementos tengan números de referencia únicos, de lo contrario no podrá distinguir los elementos. Para mayor información, consulte la descripción del comando **APPEND TO LIST**.*

*Sin importar que elemento borre, debe especificar el parámetro opcional * para permitir a 4D borrar automáticamente la sublista asociada al elemento, si la hay. Si no especifica el parámetro *, es una buena idea obtener previamente el número de referencia de la sublista (si la hay) asociada al elemento, de manera que puede borrarla, si es necesario, utilizando el comando **CLEAR LIST**.*

Ejemplo

El siguiente código borra el elemento seleccionado de la lista hList. Si el elemento tiene una sublista asociada se borra (como también toda sub-sublista):

```
DELETE FROM LIST(hList;*;*)
```

Find in list

Find in list ({* ;} lista ; valor ; alcance {; arrayElem {; *}}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de la lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
valor	Cadena	➡	Valor a buscar
alcance	Entero	➡	0=Lista principal, 1=Sublista
arrayElem	Array entero largo	⬅	- Si se omite el 2do *: array de posiciones de los elementos encontrados - Si se pasa el segundo *: array de números de referencia de los elementos encontrados
*	Operador	➡	- Si se omite: utiliza la posición de los elementos - Si se pasa: utiliza el número de referencia de los elementos
Resultado	Entero largo	↩	- Si se omite el 2do *: posición del elemento encontrado - Si se pasa 2do *: número de referencia del elemento encontrado

Descripción

*El comando **Find in list** devuelve la posición o referencia del primer elemento de la lista que es equivalente a la cadena pasada en valor. Si se encuentran varios elementos, la función también puede llenar un array arrayElem con la posición o la referencia de cada elemento.*

*Si pasa el parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de la lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza una sola representación de lista o trabaja con números de referencia de elementos (se omite el segundo *), puede utilizar indiferentemente una u otra sintaxis. En cambio, si utiliza varias representaciones de la misma lista y trabaja con posiciones de elementos (se pasa el segundo *), la sintaxis basada en el nombre del objeto es necesaria ya que la posición de los elementos puede variar de una representación a otra.*

***Nota:** si utiliza el carácter @ en el nombre del objeto de la lista y el formulario contiene varias listas que corresponden a este nombre, el comando **Find in list** se aplicará al primer objeto cuyo nombre corresponda.*

*El segundo parámetro * permite indicar si quiere trabajar con las posiciones actuales de los elementos (en ese caso, este parámetro se omite) o con las referencias absolutas de los elementos (en ese caso, debe pasarse).*

Pase en valor la cadena de caracteres a buscar. La búsqueda será del tipo "exacta"; en otras palabras, la búsqueda de "piedra" no encontrará "piedras". Sin embargo, puede utilizar el carácter arroba (@) para definir las búsquedas de tipo "comienza por," "termina en" o "contiene".

El parámetro alcance se utiliza para definir si la búsqueda debe ser llevada a cabo únicamente en el primer nivel de la lista o si debe incluir todas las sublistas. Pase 0 para limitar la búsqueda al primer nivel de la lista y 1 para extenderla búsqueda a todas las sublistas.

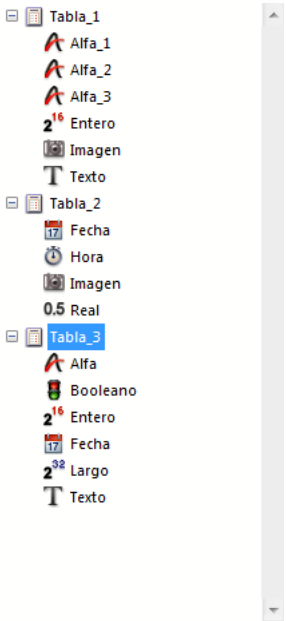
*Si quiere conocer la posición o el número de todos los elementos correspondientes a valor, pase un array entero largo en el parámetro opcional arrayElem. Si es necesario, el array será creado y redimensionado por el comando. El comando llenará el array con las posiciones (si se omite el segundo *) o los números de referencia (si se pasa el segundo *) de los elementos encontrados.*

Las posiciones se expresan en relación al elemento superior de la lista principal, teniendo en cuenta el estado actual desplegado/contraído de la lista y de las sublistas.

Si ningún elemento corresponde al valor buscado, la función devuelve 0 y el array arrayElem se devuelve vacío.

Ejemplo

Dada la siguiente lista jerárquica:



```
$vlItemPos:=Find in list(hList;"l@";1;$arrPos)
  `$vlItemPos igual a 6
  `$arrPos{1} igual a 6 y $arrPos{2} igual a 11
$vlItemRef:=Find in list(hList;"l@";1;$arrRefs;*)
  `$vlItemRef igual a 7
  `$arrRefs{1} igual a 7 y $arrRefs{2} igual a 18
$vlItemPos:=Find in list(hList;"Fecha";1;$arrPos)
  `$vlItemPos igual a 9
  `$arrPos{1} igual a 9 y $arrPos{2} igual a 16
$vlItemRefFind in list(hList;"Fecha";1;$arrRefs;*)
  `$vlItemRef igual a11
  `$arrRefs{1} igual a 11 and $arrRefs{2} igual a 23
$vlItemPos:=Find in list(hList;"Fecha";0;*)
  `$vlItemPos igual a 0
```

GET LIST ITEM

GET LIST ITEM ({* ;} lista ; posicionElem | * ; refElem ; textoElem {; sublista ; desplegada})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
posicionElem | *	Operador, Entero largo	➡	Posición del elemento en lista(s) desplegada(s) o contraída(s) * para el elemento actual de la lista.
refElem	Entero largo	⬅	Número de referencia del elemento
textoElem	Cadena	⬅	Texto del elemento de la lista
sublista	ListRef	⬅	Número de referencia de la sublista (si la hay)
desplegada	Booleano	⬅	Si una sublista está asociada: TRUE = la sublista está desplegada FALSE = la sublista está contraída

Descripción

*El comando **GET LIST ITEM** devuelve la información sobre el elemento especificado por posicionElem de la lista cuyo número de referencia o nombre de objeto se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza sólo una representación de lista, puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si usted utiliza varias representaciones de una misma lista, la lista basada en el nombre del objeto es necesaria ya que cada representación puede tener su propia configuración desplegada/contraída y su propio elemento actual.*

***Nota:** si utiliza el carácter @ en el nombre de la lista y el formulario contiene varias listas que responden a este nombre, el comando **GET LIST ITEM** sólo aplicará al primer objeto cuyo nombre corresponda.*

*La posición debe expresarse respecto al estado actual desplegado/contraído de la lista y de su sublista. Debe pasar un valor de posición entre 1 y el valor devuelto por **Count list items**. Si pasa un valor que no está en este rango, **GET LIST ITEM** devuelve valores vacíos (0, "", etc.).*

Después de la llamada, recupera:

- *El número de referencia del elemento en refElem.*
- *El texto del elemento en textElem.*

Si pasa los parámetros opcionales sublista y desplegada:

- *subLista devuelve el número de referencia de la sublista asociada al elemento. Si el elemento no tiene sublista, subLista devuelve cero (0).*
- *Si el elemento tiene una sublista, desplegada devuelve TRUE si la sublista está desplegada, y FALSE si está contraída.*

Ejemplo 1

hList es una lista cuyos elementos tienen números de referencia únicos. El siguiente código pasa por programación al estado desplegado/contraído de la sublista, si hay, asociada al elemento seleccionado:

```
$vlItemPos:=Selected list items(hList)
If($vlItemPos>0)
   GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText;$hSublist;$vbDesplegada)
   If(Is a list($hSublist))
      SET LIST ITEM(hList;$vlItemRef;$vsItemText;$vlItemRef;$hSublist;Not($vbDesplegada))
   End if
End if
```

Ejemplo 2

*Consulte el ejemplo del comando **APPEND TO LIST**.*

Get list item font

Get list item font ({* ;} lista ; refElem | *) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre de objeto de tipo lista (si se pasa *)
refElem | *	Entero largo, Operador	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista o * para el elemento actual de la lista
Resultado	Cadena	↩	Nombre de fuente

Descripción

*El comando **Get list item font** devuelve el nombre de la fuente del carácter actual del elemento especificado por el parámetro **RefElem** de la lista cuyo número de referencia o nombre de objeto se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de la lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza una sola representación de lista o trabaja con elementos estructurales (el segundo * se omite), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de una misma lista y trabaja con el elemento actual (se pasa el segundo *), la sintaxis basada en el nombre del objeto se necesita ya que cada representación puede tener su propio elemento actual.*

***Nota:** si utiliza el carácter @ en el nombre de objeto de la lista y el formulario contiene varias listas que corresponden a este nombre, el **Get list item font** se aplicará al primer objeto cuyo nombre corresponda.*

*Puede pasar un número de referencia en refElem. Si este número no corresponde a ningún elemento de la lista, el comando no hace nada. También puede pasar 0 en refElem para obtener la fuente del último elemento añadido a la lista (utilizando **APPEND TO LIST**).*

*Finalmente, puede pasar * en refElem: en este caso, el comando se aplicará al elemento actual de la lista. Si varios elementos se seleccionan manualmente, el elemento actual es el último seleccionado. Si ningún elemento está seleccionado, el comando no hace nada.*

GET LIST ITEM ICON

GET LIST ITEM ICON ({* ;} lista ; refElem | * ; icono)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre de objeto de tipo de lista (si pasa *)
refElem | *	Operador, Entero largo	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista o * para el elemento actual de la lista
icono	Variable imagen	⬅	Icono asociado al elemento

Descripción

*El comando **GET LIST ITEM ICON** devuelve, en icono, el icono asociado al elemento cuyo número de referencia se pasa en refElem de la lista cuyo número de referencia o nombre de objeto pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de la lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza una sola representación de lista o trabaja con los elementos estructurales (se omite el segundo *), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de la misma lista y trabaja con el elemento actual (se pasa el segundo *), la sintaxis basada en el nombre del objeto es requerida ya que cada representación tiene su propio elemento actual.*

***Nota:** si utiliza el carácter @ en el nombre de objeto de la lista y el formulario contiene varias listas que responden a este nombre, el comando **GET LIST ITEM ICON** se aplicará al primer objeto cuyo nombre corresponda.*

*Puede pasar un número de referencia en refElem. Si este número no corresponde a un elemento en la lista, el comando no hace nada. También puede pasar 0 en refElem para indicar el último elemento añadido a la lista (usando **APPEND TO LIST**).*

*Finalmente, puede pasar * en refElem: en este caso, el comando se aplicará al elemento actual de la lista. Si varios elementos son seleccionados manualmente, el elemento actual es el seleccionado de último. Si ningún elemento está seleccionado, el comando no hace nada.*

Pase en icono una variable imagen. Después de ejecutar el comando, contendrá el icono asociado con el elemento, sin importar la fuente del icono (imagen estática, recurso o expresión imagen).

Si ningún icono está asociado al elemento, la variable icono se devuelve vacía.

Nota:** cuando el icono asociado a un elemento ha sido definido vía una referencia estática (referencias de recursos o imágenes de la librería de imágenes), es posible conocer su número utilizando el comando **GET LIST ITEM PROPERTIES.

GET LIST ITEM PARAMETER

GET LIST ITEM PARAMETER ({* ;} lista ; refElem | * ; selector ; valor)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de la lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
refElem | *	Entero largo, Operador	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista o * para el elemento actual de la lista
selector	Cadena	➡	Constante del parámetro
valor	Cadena, Booleano, Real	⬅	Valor actual del parámetro

Descripción

El comando GET LIST ITEM PARAMETER permite conocer el valor actual del parámetro selector para el elemento refElem de la lista jerárquica cuya referencia o nombre de objeto se pasa en el parámetro lista.

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de la lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza una sola representación de lista o trabaja con los elementos estructurales (se omite el segundo *), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de una misma lista y se pasa el segundo *, es requerida la sintaxis basada en el nombre del objeto ya que cada representación puede tener su propio elemento actual.*

***Nota:** si utiliza el carácter @ en el nombre del objeto de la lisa y el formulario contiene varias listas que responden a este nombre, el comando GET LIST ITEM PARAMETER se aplicará al primer objeto cuyo nombre corresponda.*

*Puede pasar un número de referencia en refElem. Si este número no corresponde a ningún elemento de la lista, el comando no hace nada. Igualmente puede pasar 0 en refElem para indicar el último elemento añadido a la lista (utilizando **APPEND TO LIST**).*

*Finalmente, puede pasar * en refElem: en este caso, el comando se aplicará al elemento actual de la lista. Si se seleccionan manualmente varios elementos, el elemento actual es el último seleccionado. Si ningún elemento fue seleccionado, el comando no hace nada.*

*En selector, puede pasar la constante Additional text o Standard action (en el tema "**Listas jerárquicas**") o todo valor personalizado. Para mayor información sobre los parámetros selector y valor, consulte la descripción del comando **SET LIST ITEM PARAMETER**.*

GET LIST ITEM PARAMETER ARRAYS

GET LIST ITEM PARAMETER ARRAYS ({* ;} lista ; refElemento ; arrSelectores {; arrValores})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista o Nombre de objeto de tipo lista (si se pasa *)
refElemento	Entero largo, Operador	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista o * para el elemento actual de la lista
arrSelectores	Array texto	⬅	Array de los nombres de parámetros
arrValores	Array texto	⬅	Array de los valores de los parámetros

Descripción

*El comando **GET LIST ITEM PARAMETER ARRAYS** permite recuperar en una sola llamada el conjunto de los parámetros (así como también, opcionalmente, sus valores) asociados al elemento refElemento de la lista jerárquica cuya referencia o nombre de objeto se pasó en el parámetro lista.*

*Los parámetros asociados a los elementos permiten almacenar información adicional sobre cada elemento. Se definen con la ayuda del comando **SET LIST ITEM PARAMETER**.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (RefList). Si utiliza una sola representación de lista o trabaja con los elementos estructurales (el segundo * se omite), puede utilizar indiferentemente una u otra sintaxis. Sin embargo, si utiliza varias representaciones de una misma lista y trabaja con el elemento actual (se pasa el segundo *), debe utilizar la sintaxis basada en el nombre del objeto, ya que cada representación puede tener su propio elemento actual.*

***GET LIST ITEM PARAMETER ARRAYS** devuelve los parámetros definidos para el elemento refElemento en el array texto arrSelectores. Cuando se pasa el array texto arrValores, el comando lo utiliza para devolver los valores asociados con estos parámetros.*

arrValores debe ser un array de tipo texto. Si tiene valores asociados que no son textuales (tipo numérico o Booleano), convertidos en cadenas (True="1", False="0").

Ejemplo

Dada la siguiente lista jerárquica:

```
<>HL:=New list
$ID:=30
APPEND TO LIST(<>HL;"Martin";$ID)
 //5 parámetros
SET LIST ITEM PARAMETER(<>HL;$ID;"Nombre";"Phil")
SET LIST ITEM PARAMETER(<>HL;$ID;"Fecha de nacimiento";"01/02/1978")
SET LIST ITEM PARAMETER(<>HL;$ID;"Hombre";True) //Booleano
SET LIST ITEM PARAMETER(<>HL;$ID;"Edad";33) //número
SET LIST ITEM PARAMETER(<>HL;$ID;"Ciudad";"Dallas")
```

Para mayor simplicidad la lista se asoció a una lista objeto con el mismo nombre ("<>HL").

Cuando el elemento "Martin" se selecciona en la lista, puede recuperar sus parámetros ejecutando el siguiente código:

```
ARRAY TEXT(arrParamNames;0)
GET LIST ITEM PARAMETER ARRAYS(*;"<>HL";arrParamNames)
 // arrParamNames{1} contiene "Nombre"
 // arrParamNames{2} contiene "Fecha de nacimiento"
 // arrParamNames{3} contiene "Hombre"
 // arrParamNames{4} contiene "Edad"
 // arrParamNames{5} contiene "Ciudad"
```

Si también quiere obtener los valores de los parámetros, escriba:

```
ARRAY TEXT(arrParamNames;0)
ARRAY TEXT(arrParamValues;0)
GET LIST ITEM PARAMETER ARRAYS(*;"<>HL";arrParamNames;arrParamValues)
 // arrParamValues{1} contiene "Phil"
 // arrParamValues{2} contiene "01/02/1978"
 // arrParamValues{3} contiene "1"
 // arrParamValues{4} contiene "33"
 // arrParamValues{5} contiene "Dallas"
```

GET LIST ITEM PROPERTIES

GET LIST ITEM PROPERTIES ({* ;} lista ; refElem | * ; editable {; estilos {; icono {; color}}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es una referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
refElem | *	Operador, Entero largo	➡	Número de referencia del elemento, o 0 para el último elemento añadido a la lista, o * para el elemento actual de la lista
editable	Booleano	⬅	TRUE = Editable, FALSE = No editable
estilos	Entero largo	⬅	Estilo de fuente del elemento
icono	Entero largo	⬅	Número de recurso Mac OS 'cicn', o 65536 + número de recurso Mac OS 'PICT', o 131072 + número de referencia de imagen
color	Entero largo	⬅	Valor de color RGB

Descripción

*El comando **GET LIST ITEM PROPERTIES** devuelve las propiedades del elemento designado por el parámetro refElem de la lista cuyo número de referencia o nombre de objeto se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza sólo una representación de lista, puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de una misma lista y trabaja con el elemento actual (el segundo * se pasa), la sintaxis basada en el nombre del objeto es necesaria ya que cada representación puede tener su propio elemento actual.*

***Nota**: si utiliza el carácter @ en el nombre de la lista y el formulario contiene varias listas que responden a este nombre, el comando **GET LIST ITEM PROPERTIES** se aplicará al primer objeto cuyo nombre corresponde.*

*En refElem, puede pasar un número de referencia, o el valor 0 con el fin de designar el último elemento añadido a la lista, o * para designar el elemento actual de la lista. Si se seleccionan varios elementos, el elemento actual es el último en ser seleccionado.*

*Si pasa * y ningún elemento es seleccionado o si el número de referencia del elemento no corresponde a ningún elemento de la lista, el comando deja los parámetros sin cambios.*

*Si trabaja con números de referencia de los elementos, asegúrese de utilizar, construya una lista en la cual los elemento tengan números de referencia únicos, de lo contrario no podrá diferenciar los elementos. Para mayor información, consulte la descripción del comando **APPEND TO LIST**.*

Después de la llamar:

- *editable devuelve TRUE si el elemento es editable.*
- *estilos devuelve el estilo de fuente del elemento.*
- *icono devuelve el icono o la imagen asociada al elemento, 0 si no hay.*
- *color devuelve el color del texto del elemento especificado.*

***Nota:** puede recuperar, en una variable imagen, el icono asociado con un elemento utilizando el comando **GET LIST ITEM ICON**.*

*Para mayor información sobre estas propiedades, consulte la descripción del comando **SET LIST ITEM PROPERTIES.***

GET LIST PROPERTIES

GET LIST PROPERTIES (lista ; apariencia {; icono {; altoLinea {; dobleClic {; multiSeleccion {; editable}}}}})

Parámetro	Tipo		Descripción
lista	ListRef	➡	Número de referencia de la lista
apariencia	Entero largo	⬅	Estilo gráfico de la lista 1 = Lista jerárquica a la Macintosh 2 = Lista jerárquica a la Windows
icono	Entero largo	⬅	Referencia de recurso Mac OS 'cicn'
altoLinea	Entero largo	⬅	Altura mínima de la línea expresada en píxeles
dobleClic	Entero largo	⬅	Desplegar/Contraer sublista con doble-clik? 0 = Sí, 1= No
multiSeleccion	Entero largo	⬅	Selecciones múltiples: 0 = No, 1 = Sí
editable	Entero largo	⬅	Lista editable por el usuario: 0 = No, 1 = Sí

Descripción

*El comando **GET LIST PROPERTIES** devuelve información sobe la lista cuyo número de referencia se pasa en lista.*

El parámetro apariencia devuelve el estilo gráfico de la lista.
El parámetro icono devuelve las referencias de los recursos de los iconos de nodos utilizados en la lista.
El parámetro alturaLinea devuelve la altura de línea mínima.
Si dobleClic vale 1, hacer doble-clic en una elemento padre de la lista no hace que su lista hijo se despliegue o se contraiga. Si dobleClic vale 0, este comportamiento se activa (valor por defecto).
Si el parámetro multiSeleccion vale 0, las selecciones múltiples de elementos (manualmente o por programación) no son posibles en la lista. Si vale 1, se permiten las selecciones múltiples.
Si el parámetro editable vale 1, la lista es editable cuando se muestra en forma de lista de selección en los registros. Si vale 0, la lista no es editable.

*Estas propiedades pueden definirse con la ayuda del comando **SET LIST PROPERTIES** y/o en el editor de listas en el entorno Diseño, si la lista fue creada en el editor o guardada utilizando el comando **SAVE LIST**.*

*Para una completa descripción de la apariencia, iconos de nodos, altura de línea mínima y administración de una lista con doble clic, consulte el comando **SET LIST PROPERTIES**.*

INSERT IN LIST

INSERT IN LIST ({* ;} lista ; antesElem | * ; textoElem ; refElem {; sublista ; desplegada})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
antesElem | *	Entero largo, Operador	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista o * para el elemento de la lista actualmente seleccionado
textoElem	Cadena	➡	Texto para el nuevo elemento (max. 255 caracteres)
refElem	Entero largo	➡	Número de referencia único del nuevo elemento
sublista	ListRef	➡	Sublista opcional para asociar al nuevo elemento
desplegada	Booleano	➡	Indica si la sublista será desplegada o contraída

Descripción

El comando INSERT IN LIST inserta el elemento designado por el parámetro refElem en la lista cuyo número de referencia se pasa en lista.

Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de la lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (RefLista). Si utiliza una sola representación de lista o trabaja con los elementos estructurales (se omite el segundo *), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de la misma lista y trabaja con el elemento actual (se pasa el segundo *), la sintaxis basada en el nombre del objeto es requerida ya que cada representación tiene su propio elemento actual.

El parámetro antesElem puede utilizarse para designar el elemento delante del cual usted quiere insertar el nuevo elemento:

- Puede pasar el valor 0 con el fin de designar el último elemento añadido a la lista. El nuevo elemento insertado se convertirá entonces en el elemento seleccionado.
- Puede pasar * para que el nuevo elemento sea insertado antes del elemento seleccionado actualmente en la lista. En este caso, el nuevo elemento insertado se convierte en el elemento seleccionado.
- Por otra parte, si quiere insertar un elemento antes de un elemento específico, pase el número de referencia de ese elemento. En este caso, el nuevo elemento insertado no es seleccionado automáticamente. Si no hay un elemento con el número de referencia correspondiente, el comando no hace nada.

Pase el número de referencia del nuevo elemento en refElem. Aunque calificamos este número de referencia de elemento como único, realmente puede pasar el valor que desea. Consulte la sección **Gestión de listas jerárquicas** para obtener más información sobre el parámetro refElem.

Si quiere que el elemento incluya subelementos, pase un número de referencia de lista válido en el parámetro sublista. En este caso, también debe pasar el parámetro desplegada. Pase **True** o **False** en este parámetro de manera que esta sublista se muestre desplegada o contraída respectivamente.

Ejemplo

El siguiente código inserta un elemento (sin sublista asociada) justo antes del elemento seleccionado actualmente en la lista hList:

```
vlUniqueRef:=vlUniqueRef+1
INSERT IN LIST(hList;*;"Nuevo elemento";vlUniqueRef)
```

Is a list

Is a list (lista) -> Resultado

Parámetro	*Tipo*		*Descripción*
lista	*ListRef*	⇒	*Referencia de la lista a probar*
Resultado	*Booleano*	⇐	*TRUE si la lista es una lista jerárquica FALSE si la lista no es una lista jerárquica*

Descripción

*El comando **Is a list** devuelve TRUE si el valor pasado en lista es una referencia válida a una lista jerárquica. De lo contrario, devuelve FALSE.*

Ejemplo 1

*Ver el ejemplo del comando **CLEAR LIST**.*

Ejemplo 2

*Ver los ejemplos del comando **DRAG AND DROP PROPERTIES**.*

List item parent

List item parent ({* ;} lista ; refElem | *) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de la lista (si se omite *), o Nombre de objeto de tipo lista (si se pasa*)
refElem | *	Operador, Entero largo	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista o * para el elemento actual de la lista
Resultado	Entero largo	↩	Número de referencia del elemento padre o 0 si no hay

Descripción

*El comando **List item parent** devuelve el número de referencia del elemento padre.*

Pase en lista el número de referencia o el nombre de objeto de la lista.

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza sólo una representación de lista o trabaja con elementos estructurales (el segundo * se omite), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de la misma lista y trabaja con el elemento actual (el segundo * es pasado), la sintaxis basada en el nombre del objeto es necesaria ya que cada representación tiene su propio elemento actual.*

***Nota**: si utiliza el carácter @ en el nombre del objeto de la lista y el formulario contiene varias listas que tienen este nombre, el comando **List item parent** se aplicará al primer objeto cuyo nombre corresponda.*

*Pase en refElem un número de referencia de elemento en la lista ó 0 ó *. Si pasa 0, el comando aplica al último elemento añadido a la lista. Si pasa *, el comando aplica el elemento actual de la lista. Si varios elementos han sido seleccionados manualmente, el elemento actual es el último elemento seleccionado.*

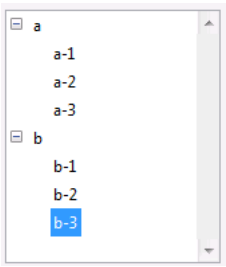
En cambio, si el elemento correspondiente existe en la lista y si este elemento está en una sublista (y por lo tanto tiene un elemento padre), usted obtiene el número de referencia del elemento padre.

*Si no existe un elemento con el número de referencia que pasó, o si pasó * y ningún elemento está seleccionado, o si el elemento no tiene padre, **List item parent** devuelve 0 (cero).*

*Si trabaja con números de referencia de los elementos, asegúrese de construir una lista en la cual los elementos tengan números de referencia únicos; de lo contrario no podrá diferenciar los elementos. Para mayor información, consulte la descripción del comando **APPEND TO LIST**.*

Ejemplo

He aquí una lista llamada hList mostrada en el entorno Aplicación:



Los números de referencia de los elementos son los siguientes:

Elemento	Número
a	100
a - 1	101
a - 2	102
a - 3	103
b	200
b - 1	201
b - 2	202
b - 3	203

- *En el siguiente código, si se selecciona el elemento "b - 3", la variable $vlPadreElemRef toma el valor 200, es decir el número de referencia del elemento "b":*

```
$vlItemPos:=Selected list items(hList)
GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText)
$vlPadreElemRef :=List item parent(hList;$vlItemRef) ` $vlPadreElemRef vale 200
```

- *Si se selecciona el elemento "a - 1", la variable $vlPadreElemRef toma el valor 100, es decir el número de referencia del elemento "a".*

- *Si se selecciona el elemento el elemento "a" o "b", la variable $vlPadreElemRef toma el valor 0, porque estos elementos no tienen elemento padre.*

List item position

List item position ({* ;} lista ; refElem) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es una referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) Nombre de objeto de tipo lista (si se pasa *)
refElem	Entero largo	➡	Número de referencia del elemento
Resultado	Entero largo	↩	Posición del elemento en listas desplegadas/contraídas

Descripción

*El comando **List item position** devuelve la posición del elemento cuyo número de referencia o nombre de objeto se pasa en refElem, en la lista cuyo número de referencia se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza sólo una representación de lista o trabaja con elementos estructurales (el segundo * se omite), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de la misma lista, la sintaxis basada en el nombre del objeto es necesaria ya que cada representación puede tener su propia configuración desplegada/contraída.*

***Nota:** si utiliza el carácter @ en el nombre del objeto de la lista y el formulario contiene varias listas que tienen este nombre, el comando **List item position** se aplicará al primer objeto cuyo nombre corresponda.*

***Nota:** a diferencia de otros comandos de este tema, este comando no permite pasar el valor 0 en refElem para designar el último elemento añadido.*

La posición se expresa con relación al elemento superior de la lista, utilizando, utilizando el estado actual desplegado/contraído de la lista y su sublista.

*El resultado es un número entre 1 y el valor devuelto por **Count list items**.*

*Si el elemento no es visible porque está ubicado en una lista contraída, **List item position** despliega la lista correspondiente de manera que el elemento sea visible.*

*Si el elemento no existe, **List item position** devuelve 0.*

LIST OF CHOICE LISTS

LIST OF CHOICE LISTS (arrayNums ; arrayNoms)

Parámetro	***Tipo***		***Descripción***
arrayNums	*Array entero largo*	⇐	*Números de las listas*
arrayNoms	*Array texto*	⇐	*Nombres de las listas*

Descripción

*El comando **LIST OF CHOICE LISTS** devuelve, en los arrays sincronizados arraysNums y arrayNoms, los números y los nombres de las listas definidas en el editor de listas en el entorno Diseño.*

Los números de las listas corresponden a su orden de creación. En el editor de listas, las listas se muestran en orden alfabético.

Load list

Load list (nomLista) -> Resultado

Parámetro	Tipo		Descripción
nomLista	Cadena	➡	Nombre de una lista creada en el Editor de listas del entorno Diseño
Resultado	ListRef	⮌	Número de referencia de la lista creada recientemente

Descripción

Load list *crea una lista jerárquica cuyo contenido se copia de la lista pasada en nomLista. Luego devuelve el número de referencia de la lista creada recientemente.*

Para asegurarse de que la lista especificada por nomLista existe, utilice la función ***Is a list****.*

Note que la nueva lista es una copia de la lista definida en el entorno Diseño. Por lo tanto, cualquier modificación realizada a la nueva lista no afectará la lista definida en el entorno Diseño. Igualmente, toda modificación posterior a la lista definida en el entorno Diseño no afectará la lista que acaba de crear.

Si modifica la lista creada recientemente y quiere guardar los cambios de forma permanente, llame al comando ***SAVE LIST****.*

Recuerde llamar ***CLEAR LIST*** *para borrar la lista creada cuando haya terminado. De lo contrario, permanecerá en memoria hasta el final de la sesión de trabajo o hasta que el proceso en el cual fue creada termine o sea abortado.*

Consejo: *si asocia una lista a un objeto de formulario (lista jerárquica, pestaña, o menú jerárquico) utilizando Lista de valores en la ventana de Lista de propiedades, no necesita llamar* ***Load list*** *o* ***CLEAR LIST*** *en el método del objeto. 4D carga y borra la lista automáticamente por usted.*

Ejemplo

Usted crea una base para el mercado internacional y necesita cambiar a los diferentes idiomas mientras utiliza la base. En un formulario, presenta una lista jerárquica, llamada hlList, que ofrece una lista de opciones estándar. En el entorno Diseño, usted preparó varias listas, tales como "Opciones EN" para la versión en inglés, "Opciones FR" para la versión en francés, "Opciones ES" para la versión en español, etc. Adicionalmente, usted mantiene una variable interproceso llamada◇gsIdiomaActual, donde almacena un código de lenguaje de 2 caracteres, como "EN" para la versión en inglés, "FR" para la versión en francés, "ES" para la versión en español, etc. Para asegurarse de que se cargue la lista correcta utilizando el idioma seleccionado actualmente, puede escribir:

```
` Método de objeto de la lista jerárquica hlList
Case of
   :(Form event=On Load)
      C_LONGINT(hlList)
      hlList:=Load list("Std Options"+◇gsIdiomaActual)
   :(Form event=On Unload)
      CLEAR LIST(hlList;*)
End case
```

New list

New list -> Resultado

Parámetro	Tipo	Descripción
Resultado	ListRef	Número de referencia de lista

Descripción

New list *crea una nueva lista jerárquica vacía en memoria y devuelve su número de referencia único.*
Advertencia: *las listas jerárquicas residen en la memoria. Cuando terminé de utilizar una lista jerárquica, es importante borrarla para liberar memoria, utilizando el comando* ***CLEAR LIST****.*
Otros comandos le permiten crear listas jerárquicas:

- ***Copy list*** *duplica una lista existente.*
- ***Load list*** *crea una nueva lista cargando una lista creada (manualmente o por programación) en el editor de listas del entorno Diseño.*
- ***BLOB to list*** *crea una lista a partir del contenido de un BLOB en el cual una lista fue previamente guardada.*

Una vez haya creado una lista jerárquica utilizando ***New list****, puede:*

- *Añadir elementos a esa lista, utilizando el comando* ***APPEND TO LIST*** *o* ***INSERT LIST ITEM****.*
- *Borrar elementos de esa lista, utilizando el comando* ***DELETE FROM LIST****.*

Ejemplo

Ver el ejemplo del comando ***APPEND TO LIST****.*

SAVE LIST

SAVE LIST (lista ; nomLista)

Parámetro	Tipo		Descripción
lista	ListRef	➡	Número de referencia de la lista
nomLista	Cadena	➡	Nombre de la lista como aparecerá en el editor de listas del entorno Diseño

Descripción

*El comando **SAVE LIST** guarda la lista cuyo número de referencia pasó en lista, en el editor de listas en el entorno Diseño, bajo el nombre que pasó en nomLista.*

Su contenido se reemplazará si ya hay una lista con el mismo nombre.

SELECT LIST ITEMS BY POSITION

SELECT LIST ITEMS BY POSITION ({* ;} lista ; posicionElem {; posicionArray})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
posicionElem	Entero largo	➡	Posición del elemento en la(s) lista(s) desplegada(s) /contraída(s)
posicionArray	Array entero largo	➡	Array de posiciones en la(s) lista(s) desplegada(s) /contraída(s)

Descripción

*El comando **SELECT LIST ITEMS BY POSITION** selecciona el(los) elemento(s) cuya posición se pasa en posicionElem y opcionalmente en posicionArray en la lista cuyo número de referencia o nombre de objeto se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza sólo una representación de lista, puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si usted utiliza varias representaciones de una misma lista, la lista basada en el nombre del objeto es necesaria ya que cada representación puede tener su propia configuración desplegada/contraída.*

***Nota:** si utiliza el carácter @ en el nombre de la lista y el formulario contiene varias listas que responden a este nombre, el comando **SELECT LIST ITEMS BY POSITION** sólo aplicará al primer objeto cuyo nombre corresponde.*

*La posición de los elementos siempre se expresa utilizando el estado desplegado/contraído de la lista y sus sublistas. Usted pasa un valor de posición entre 1 y el valor devuelto por **Count list items**. Si pasa un valor fuera de este rango, no se selecciona ningún elemento.*

Si no pasa el parámetro posicionArray, el parámetro posicionElem representa la posición del elemento a seleccionar.

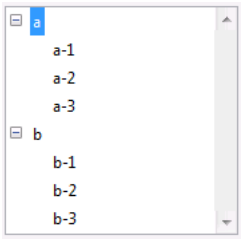
El parámetro opcional posicionArray le permite seleccionar varios elementos simultáneamente de la lista. En posicionArray, debe pasar un array donde cada línea indique la posición de un elemento a seleccionar.

*Cuando pasa este parámetro, el elemento designado por el parámetro posicionElem designa el nuevo elemento actual de la lista en la selección resultante, el cual puede pertenecer o no al conjunto de elementos definido por el array. El elemento actual es, más particularmente, el que pasa a modo edición si se utiliza el comando **EDIT ITEM**.*

***Nota:** para que varios elementos puedan ser seleccionados simultáneamente en una lista jerárquica (manualmente o por programación), la propiedad multi-seleccionable debe haber sido activada para la lista. Esta propiedad se define utilizando el comando **SET LIST PROPERTIES**.*

Ejemplo

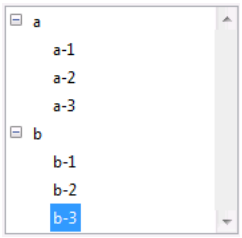
Dada la lista jerárquica llamada hList, mostrada en el entorno Aplicación:



Después de la ejecución de este código:

```
SELECT LIST ITEMS BY POSITION(hList;Count list items(hList))
```

El último elemento visible de la lista es seleccionado:

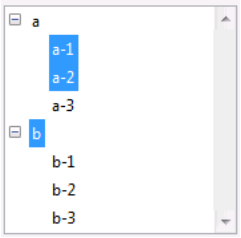


Después de la ejecución de las siguientes líneas de código:

```
SET LIST PROPERTIES(hList;0;0;18;0;1)
  `Es imperativo pasar 1 como último parámetro para permitir las selecciones múltiples
ARRAY LONGINT($arr;3)
$arr{1}:=2
$arr{2}:=3
$arr{3}:=5
```

```
SELECT LIST ITEMS BY POSITION(hList;3;$arr)
  `El tercer elemento se designa como elemento actual
```

Se seleccionan los elementos segundo, tercero y quinto de la lista jerárquica



SELECT LIST ITEMS BY REFERENCE

SELECT LIST ITEMS BY REFERENCE (lista ; refElem {; refArray})

Parámetro	Tipo		Descripción
lista	ListRef	➡	Número de referencia de lista
refElem	Entero largo	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista
refArray	Array entero largo	➡	Array de números de referencia de elementos

Descripción

*El comando **SELECT LIST ITEMS BY REFERENCE** selecciona el o los elementos cuyo número de referencia se pasa en refElem y opcionalmente en refArray, en la lista cuyo número de referencia se pasa en lista.*

Si ningún elemento tiene el número de referencia que pasó, el comando no hace nada.

Si un elemento no es visible actualmente (por ejemplo, si está ubicado en una lista contraída), el comando despliega la(s) sublista(s) correspondiente(s) de manera que quede(n) visible(s).

Si no pasa el parámetro refArray, el parámetro refElem representa la referencia del elemento a seleccionar. Si el número de elemento no corresponde a ningún elemento de la lista, el comando no hace nada. Igualmente puede pasar el valor 0 en este parámetro para designar el último elemento añadido a la lista.

El parámetro opcional refArray le permite seleccionar varios elementos simultáneamente en la lista. En refArray, debe pasar un array donde cada línea indique la referencia fija de un elemento a seleccionar.
*En este caso, el elemento designado por el parámetro refElemf determina el nuevo elemento actual de la lista en la selección resultante, el cual puede pertenecer o no al conjunto de elementos definido por el array. El elemento actual es, más particularmente, el que es editado por el comando **EDIT ITEM**.*

***Nota:** para seleccionar varios elementos simultáneamente en una lista jerárquica (manualmente o por programación), la propiedad multi-seleccionable debe haber sido seleccionada para la lista. Esta propiedad se define utilizando el comando **SET LIST PROPERTIES**.*

*Si trabaja con los números de referencia de los elementos, asegúrese de construir un alista en la cual los elementos tengan números de referencia únicos; de lo contrario no podrá diferenciarlos. Para mayor información, consulte la descripción del comando **APPEND TO LIST**.*

Ejemplo

hList es una lista cuyos elementos tienen números de referencia únicos. El siguiente método de objeto para un botón selecciona el elemento padre (si lo hay) del elemento seleccionado actualmente:

```
$vlItemPos:=Selected list items(hList)  ` Obtener la posición del elemento seleccionado
GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText)  ` Obtener el número de referencia del elemento seleccionado
$vlParentItemRef:=List item parent(hList;$vlItemRef)  ` Obtener número de referencia del elemento padre (si lo hay)
If($vlParentItemRef>0)
   SELECT LIST ITEM BY REFERENCE(hList;List item parent(hList;$vlItemRef))  ` Selección del elemento padre End if
```

Selected list items

Selected list items ({* ;} lista {; arrayElem {; *}}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica: lista es un nombre de objeto (cadena) Si se omite, lista es una referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto tipo lista (si se pasa *)
arrayElem	Array entero largo	⬅	Si se omite el segundo *: Array contiene las posiciones de los elementos seleccionados en la(s) lista(s) Si se pasa el segundo *: Array contiene las referencias de los elementos seleccionados
*	Operador	➡	Si se omite: posición(es) de(l) (los) elemento(s) Si se pasa: referencia(s) de(l) (los) elemento(s)
Resultado	Entero largo	↩	Si se omite segundo *: posición del elemento seleccionado en la(s) lista(s) desplegada(s) Si se pasa el segundo *: Referencia del elemento seleccionado

Descripción

*El comando **Selected list items** devuelve la posición o referencia del elemento seleccionado en la lista cuyo número de referencia o nombre de objeto se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza sólo una representación de lista o trabaja con referencias de elementos (el segundo * se pasa), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de una misma lista y trabaja con posiciones de elementos (el segundo * se omite), la sintaxis basada en el nombre del objeto es necesaria ya que cada representación puede tener su propia configuración de elementos desplegados/contraídos.*

***Nota**: si utiliza el carácter @ en el nombre del objeto de la lista y el formulario contiene varias listas que responden a este nombre, el comando **Selected list items** sólo aplicará al primer objeto cuyo nombre corresponde.*

En el caso de selección múltiple, el comando puede devolver también en el array elemArray, la posición o referencia de cada elemento seleccionado. Esta función debe ser aplicada a una lista mostrada en un formulario con el fin de detectar el (los) elementos seleccionado(s) por el usuario.

*El segundo parámetro * le permite indicar si quiere trabajar con las posiciones actuales de los elementos (en este caso, se debe omitir el parámetro *) o con referencias fijas de los elementos (en este caso, debe utilizarse el parámetro *).*

Puede pasar un array entero largo en el parámetro elemArray. Si es necesario, el array será creado y redimensionado por el comando. Una vez ejecutado el comando, elemArray contendrá:

- *la posición de cada elemento seleccionado relativa al estado contraído/desplegado de la(s) lista(s) desplegada(s) si se omite el parámetro *.*
- *la referencia fija de cada elemento seleccionado si se pasa el parámetro *.*
  Si no se han seleccionado elementos, el array se devuelve vacío.

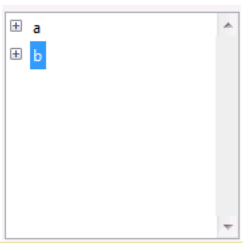
***Nota:** en caso de selección múltiple, el comando devuelve la posición o referencia del elemento actual de lista. El elemento actual es el último elemento en el que el usuario hizo clic (selección manual) o el elemento designado por los comandos **SELECT LIST ITEMS BY POSITION** o **SELECT LIST ITEMS BY REFERENCE** (selección por programación).*

Si la lista tiene sublistas, aplique el comando a la lista principal (la que está definida en el formulario), y no a una de sus sublistas. Las posiciones son expresadas con respecto al elemento superior de la lista principal, utilizando el estado actual desplegado/expandido de la lista y sus sublistas.

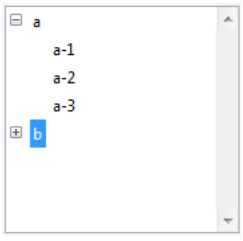
En todos los casos, si ningún elemento está seleccionado, la función devuelve 0.

Ejemplo

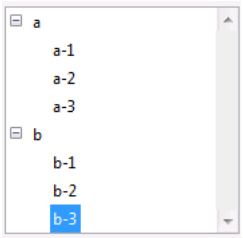
Esta es una lista llamada hList, mostrada en el entorno Aplicación:



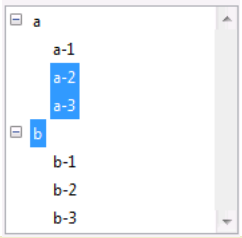

```
$vlItemPos:=Selected list items(hList) ` en este punto $vlItemPos vale 2
```



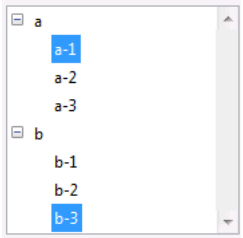
```
$vlItemPos:=Selected list items(hList) ` en este punto $vlItemPos vale 4
$vlItemRef:=Selected list items(hList;*) ` $vlItemRef vale 200 (por ejemplo)
```




```
$vlItemPos:=Selected list items(hList) ` en este punto $vlItemPos vale 8
$vlItemRef:=Selected list items(hList;*) ` $vlItemRef vale 203 (por ejemplo)
```




```
$vlItemPos:=Selected list items(hList;$arrPos) ` en este punto, $vlItemPos vale 3
 ` $arrPos{1} vale 3, $arrPos{2} vale 4 y $arrPos{3} vale 5
```




```
$vlItemRef:=Selected list items(hList;$arrRefs;*) ` $vlItemRef vale 203 (por ejemplo)
 ` $arrRefs{1} vale 101, $arrRefs{2} vale 203 (por ejemplo)
```

SET LIST ITEM

SET LIST ITEM ({* ;} lista ; refElem | * ; textElem ; nuevaRef {; sublista ; desplegada})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es una referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre de objeto de tipo lista (si se pasa*)
refElem | *	Operador, Entero largo	➡	Número de referencia de elemento, o 0 para el último elemento añadido a la lista, o * para el elemento actual de la lista
textElem	Cadena	➡	Nuevo texto del elemento
nuevaRef	Entero largo	➡	Nuevo número de referencia del elemento
sublista	ListRef	➡	Nueva sublista asociada al elemento, o 0 = ninguna sublista (separar actual, si la hay), o -1 = ningún cambio
desplegada	Booleano	➡	Indica si la sublista debe ser desplegada o contraída

Descripción

*El comando **SET LIST ITEM** modifica el elemento designado por el parámetro refElem en la lista cuyo número de referencia o nombre de objeto se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (**RefLista**). Si utiliza sólo una representación de la lista o trabaja con elementos estructurales (se omite el segundo *), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de una misma lista y trabaja con el elemento actual (el segundo * se pasa), la sintaxis basada en el nombre del objeto es necesaria ya que cada representación puede tener su propio elemento actual.*

*Puede pasar un número de referencia en refElem. Si no hay un elemento de la lista con el número de referencia que pasó, el comando no hace nada. Opcionalmente puede pasar 0 en refElem para designar el último elemento añadido a la lista utilizando **APPEND TO LIST**.*

*Finalmente, puede pasar * en refElem: en este caso, el comando se aplicará al elemento actual de la lista. Si varios elementos son seleccionados manualmente, el elemento actual es aquel que fue seleccionado de último. Si ningún elemento está seleccionado, el comando no hace nada.*

*Si trabaja con los números de referencia de los elementos, construya una lista en la cual los elementos tengan números de referencia únicos, de lo contrario no podrá diferenciar los elementos. Para mayor información, consulte la sección **Gestión de listas jerárquicas**.*

Pase el nuevo texto del elemento en textElem. Para cambiar el número de referencia del elemento, pase el nuevo valor en nuevaRef; de lo contrario, pase el mismo valor que en refElem.

Para asociar una lista a un elemento, pase el número de referencia de la sublista en subLista. En este caso, puede igualmente especificar si quiere que la nueva sublista aparezca desplegada pasando TRUE en desplegada; de lo contrario, pase FALSE.

*Para desasociar una sublista que se encuentre asociada al elemento, pase 0 (cero) en sublista. En este caso, es una buena idea haber obtenido previamente el número de referencia de esa lista utilizando **GET LIST ITEM**, de manera que pueda borrar la sublista más adelante utilizando **CLEAR LIST**, si ya no la necesita más.*

Si no quiere cambiar las propiedades de la sublista del elemento, pase -1 en sublista.

***Nota:** incluso si son opcionales, los parámetros sublista y desplegada deben pasarse de manera conjunta.*

Ejemplo 1

hList es una lista cuyos elementos tienen números de referencia únicos. El siguiente método de objeto de un botón añade un elemento hijo al elemento actualmente seleccionado en la lista.

```
$vlItemPos:=Selected list items(hList)
If($vlItemPos>0)
   GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText;$hSublist;$vbDesplegada)
   $vbNuevaSubList:=Not(Is a list($hSublist))
   If($vbNuevaSubList)
      $hSublist:=New list
   End if
   vlUniqueRef:=vlUniqueRef+1
   APPEND TO LIST($hSubList;"New Item";vlUniqueRef)
   If($vbNuevaSubList)
      SET LIST ITEM(hList;$vlItemRef;$vsItemText;$vlItemRef;$hSublist;True)
   End if
   SELECT LIST ITEMS BY REFERENCE(hList;vlUniqueRef)
End if
```

Ejemplo 2

*Ver ejemplo del comando **GET LIST ITEM**.*

Ejemplo 3

*Ver ejemplo del comando **APPEND TO LIST**.*

SET LIST ITEM FONT

SET LIST ITEM FONT ({* ;} lista ; refElem | * ; fuente)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
refElem | *	Entero largo, Operador	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista o * para el elemento actual de la lista
fuente	Cadena, Entero largo	➡	Nombre o número de fuente

Descripción

*El comando **SET LIST ITEM FONT** modifica la fuente de caracteres del elemento especificado por el parámetro refElem de la lista cuyo número de referencia o nombre de objeto se pasa en lista.*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de la lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (ListRef). Si utiliza una sola representación de lista o trabaja con los elementos estructurales (se omite el segundo *), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de la misma lista y trabaja con el elemento actual (se pasa el segundo *), la sintaxis basada en el nombre del objeto es requerida ya que cada representación puede tener su propio elemento actual.*

*Puede pasar un número de referencia en refElem. Si este número no corresponde a ningún elemento de la lista, el comando hace nada. Igualmente puede pasar 0 en refElem para solicitar la modificación del último elemento añadido a la lista (utilizando **APPEND TO LIST**).*

*Por último, puede pasar * en refElem: en este caso, el comando se aplicará al elemento actual de la lista. Si varios elementos se seleccionan manualmente, el elemento actual es el que se seleccionó de último. Si ningún elemento está seleccionado, el comando no hace nada.*

En el parámetro fuente, pase el nombre o número de la fuente a utilizar. Para reaplicar la fuente por defecto de la lista jerárquica, pase una cadena vacía en fuente.

Ejemplo

Aplicar la fuente Times al elemento actual de la lista:

```
SET LIST ITEM FONT(*;"Milista";*;"Times")
```

SET LIST ITEM ICON

SET LIST ITEM ICON ({* ;} lista ; refElem | * ; icono)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
refElem | *	Entero largo, Operador	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista o * para el elemento actual de la lista
icono	Imagen	➡	Icono a asociar al elemento

Descripción

*El comando **SET LIST ITEM ICON** permite modificar el icono asociado al elemento especificado por el parámetro refElem de la lista cuyo número de referencia o nombre de objeto se pasa en lista*

***Nota:** es posible modificar el icono asociado con un elemento utilizando el comando **SET LIST ITEM PROPERTIES**. Sin embargo, este comando acepta únicamente las referencias de imágenes estáticas (referencias de recursos o imágenes de la librería de imágenes).*

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de la lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (RefLista). Si utiliza una sola representación de lista o trabaja con los elementos estructurales (el segundo * se omite), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de la misma lista y trabaja con el elemento actual (se pasa el segundo *), se requiere la sintaxis basada en el nombre del objeto ya que cada representación puede tener su propio elemento actual.*

*Puede pasar un número de referencia en refElem. Si este número no corresponde a ningún elemento en la lista, el comando no hace nada. Igualmente puede pasar 0 en refElem para indicar el último elemento añadido a la lista (utilizando **APPEND TO LIST**).*

*Por último, puede pasar * en refElem: en este caso, el comando será aplicado al elemento actual de la lista. Si se seleccionan varios elementos manualmente, el elemento actual es el último seleccionado. Si ningún elemento está seleccionado, el comando no hace nada.*

Pase en el parámetro icono una expresión de imagen 4D válida (campo, variable, puntero, etc.). La imagen será colocada a la izquierda del elemento.

Ejemplo

Queremos asignar la misma imagen a dos elementos diferentes. El siguiente código se optimiza ya que la imagen se carga en la memoria sólo una vez:

```
C_PICTURE($picture)
READ PICTURE FILE("myPict.png";$picture)
SET LIST ITEM ICON(mylist;ref1;$picture)
SET LIST ITEM ICON(mylist;ref2;$picture)
```

SET LIST ITEM PARAMETER

SET LIST ITEM PARAMETER ({* ;} lista ; refElem | * ; selector ; valor)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es un número de referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
refElem | *	Operador, Entero largo	➡	Número de referencia del elemento o 0 para el último elemento añadido a la lista o * para el elemento actual de la lista
selector	Cadena	➡	Constante de parámetro
valor	Cadena, Booleano, Real	➡	Valor del parámetro

Descripción

El comando SET LIST ITEM PARAMETER permite modificar el parámetro selector para el elemento refElem de la lista jerárquica cuya referencia o nombre de objeto se pasa en el parámetro lista.

*Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de la lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica (refLista). Si utiliza una sola representación de lista o trabaja con los elementos estructurales (se omite el segundo *), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de la misma lista y trabaja con el elemento actual (se pasa el segundo *), se requiere la sintaxis basada en el nombre del objeto ya que cada representación puede tener su propio elemento actual.*

*Puede pasar un número de referencia en refElem. Si este número no corresponde a ningún elemento de la lista, el comando no hace nada. Puede pasar también 0 en refElem para indicar el último elemento añadido a la lista (utilizando **Listas jerárquicas**).*

*Finalmente, puede pasar * en refElem: en este caso, el comando será aplicado al elemento actual de la lista. Si se seleccionan varios elementos manualmente, el elemento actual es el último seleccionado. Si ningún elemento está seleccionado, el comando no hace nada.*

En selector, puede pasar:

- *una de las siguientes constantes (del tema "**Listas jerárquicas**"):*

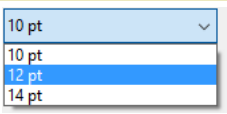
Constante	Tipo	Valor	Comentario
Additional text	Cadena	4D_additional_text	Esta constante se utiliza para agregar texto a la derecha del elemento refElem. Este título adicional siempre se mostrará en la parte derecha de la lista, incluso cuando el usuario mueva el cursor de desplazamiento horizontal. Cuando utilice esta constante, pase el texto que se mostrará en valor.
Associated standard action	Cadena	4D_standard_action_name	Asociar una acción estándar con el refElem. En este caso, debe pasar en el parámetro valor un nombre de acción estándar con un parámetro, por ejemplo "fontSize?value=10pt". Para más información, consulte la sección **Acciones estándar** del manual de Diseño.

- *o un **selector personalizado**: también puede pasar un texto personalizado y asociarlo con un valor de tipo texto, numérico o booleano. Este valor será almacenado con el elemento y podrá recuperarse utilizando el comando **GET LIST ITEM PARAMETER**. Este principio permite configurar todo tipo de interfaz asociado con las listas jerárquicas. Por ejemplo, en una lista de nombres de clientes, puede guardar la edad de cada persona y mostrarla únicamente cuando el elemento correspondiente sea seleccionado.*

Ejemplo

Usted desea definir como lista de opciones de un menú emergente jerárquico una lista personalizada de valores de tamaño de fuente utilizando la funcionalidad de acciones estándar:

```
$myList:=New list
APPEND TO LIST($myList;ak standard action title;1)
APPEND TO LIST($myList;ak standard action title;2)
APPEND TO LIST($myList;ak standard action title;3)
SET LIST ITEM PARAMETER($myList;1;Associated standard action;"fontSize?value=10pt")
SET LIST ITEM PARAMETER($myList;2;Associated standard action;"fontSize?value=12pt")
SET LIST ITEM PARAMETER($myList;3;Associated standard action;"fontSize?value=14pt")
OBJECT SET LIST BY REFERENCE(*;"popup";Choice list;$myList)
```



SET LIST ITEM PROPERTIES

SET LIST ITEM PROPERTIES ({* ;} lista ; refElem | * ; editable ; estilos ; icono {; color})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, lista es un nombre de objeto (cadena) Si se omite, lista es una referencia de lista
lista	ListRef, Cadena	➡	Número de referencia de lista (si se omite *) o Nombre del objeto de tipo lista (si se pasa *)
refElem | *	Operador, Entero largo	➡	Número de referencia del elemento, o 0 para el último elemento añadido a la lista, o * para el elemento actual de la lista
editable	Booleano	➡	TRUE = Editable, FALSE = No-editable
estilos	Entero largo	➡	Estilo de fuente para el elemento
icono	Entero largo	➡	Número de recurso Mac OS 'cicn' o 65536 + Número de recurso Mac OS 'PICT', o 131072 + número de referencia de imagen
color	Entero largo	➡	Valor de color RGB o -1 = restablecer color original

Descripción

El comando **SET LIST ITEM PROPERTIES** modifica el elemento designado por el parámetro refElem de la lista cuyo número de referencia o nombre de objeto se pasa en lista.

Si pasa el primer parámetro opcional *, indica que el parámetro lista es un nombre de objeto (cadena) correspondiente a una representación de lista en el formulario. Si no pasa este parámetro, indica que el parámetro lista es una referencia de lista jerárquica RefLista). Si utiliza sólo una representación de lista o trabaja con elementos estructurales (el segundo * se omite), puede utilizar indiferentemente una u otra sintaxis. Por el contrario, si utiliza varias representaciones de una misma lista y trabaja con el elemento actual (el segundo * se pasa), la sintaxis basada en el nombre del objeto es necesaria ya que cada representación puede tener su propio elemento actual.

Puede pasar un número de referencia en refElem. Si el número no corresponde a ningún elemento de la lista, el comando no hace nada. Opcionalmente puede pasar 0 en refElem para modificar el último elemento añadido a la lista utilizando **APPEND TO LIST**.

Por último, puede pasar * en refElem: en este caso, el comando se aplicará al elemento actual de la lista. Si varios elementos se seleccionan manualmente, el elemento actual es el que fue seleccionado de último. Si no se selecciona un elemento, el comando no hace nada.

Si trabaja con los números de referencia de los elementos, construya una lista en la cual los elementos tengan números de referencia únicos, de lo contrario no podrá diferenciar los elementos. Para mayor información, consulte la sección **Gestión de listas jerárquicas**.

Nota: para cambiar el texto del elemento o su sublista, utilice el comando **SET LIST ITEM**.

Para hacer que un elemento sea editable, pase TRUE en editable; o de lo contrario, pase FALSE.

Importante

Para que un elemento sea editable, debe pertenecer a una lista que sea editable. Para hacer que una lista sea editable, utilice el comando **OBJECT SET ENTERABLE**. Para hacer que un elemento individual sea editable, utilice **SET LIST ITEM PROPERTIES** . La modificación de la propiedad editable a nivel de la lista no afecta las propiedades individuales de cada elemento. Sin embargo, un elemento puede ser editable sólo si su lista es editable.

El estilo de fuente del elemento se especifica en el parámetro estilos. Se pasa una o una combinación de las siguientes constantes predefinidas:

Constante	Tipo	Valor
Plain	Entero largo	0
Bold	Entero largo	1
Italic	Entero largo	2
Underline	Entero largo	4

Si quiere asociar un icono al elemento, pase Use PicRef+N en el parámetro icono donde N es el número de referencia de una imagen almacenada en la librería de imágenes de 4D, en modo Diseño. Si no quiere asociar la imagen al elemento , pase cero (0) en icono.

Notas:

- Use PICT resource y Use PicRef son constantes predefinidas ubicadas en el tema **Listas jerárquicas**.
- Si quiere utilizar expresiones imagen 4D (campos, variables, etc.) para definir los iconos de los elementos, utilice el comando **SET LIST ITEM ICON**.

  El parámetro color (opcional) le permite modificar el color del texto del elemento. El color debe especificarse en forma de color RGB, es decir un entero largo de 4 bytes en formato 0x00RRGGBB. Para mayor información sobre este formato, consulte la descripción del comando **OBJECT SET RGB COLORS**. Pase -1 en el parámetro color para restablecer el color original del elemento.

Ejemplo 1

Ver el ejemplo del comando **APPEND TO LIST**.

Ejemplo 2

El siguiente ejemplo cambia el texto del elemento actual de lista a negrita y rojo vivo:

```
SET LIST ITEM PROPERTIES(list;*;True;Bold;0;0x00FF0000)
```

SET LIST PROPERTIES

SET LIST PROPERTIES (lista ; apariencia {; icono {; altoLinea {; dobleClic {; multiSeleccion {; editable}}}}})

Parámetro	Tipo		Descripción
lista	ListRef	➡	Número de referencia de la lista
apariencia	Entero largo	➡	Estilo gráfico de la lista 1 = Lista jerárquica a la Macintosh 2 = Lista jerárquica a la Windows 0 = Aparición auto dependiendo de la plataforma
icono	Entero largo	➡	ID de recurso Mac OS 'cicn' o 0 = icono por defecto de la plataforma
altoLinea	Entero largo	➡	Altura mínima de la línea expresada en píxeles
dobleClic	Entero largo	➡	Desplegar/Contraer sublista con doble-clic 0 = Sí, 1= No
multiSeleccion	Entero largo	➡	Selecciones múltiples: 0 = No (por defecto), 1 = Sí
editable	Entero largo	➡	0 = Lista no editable por el usuario, 1 = Lista editable por el usuario (por defecto)

Descripción

El comando **SET LIST PROPERTIES** define el alto de línea y el funcionamiento de las lista jerárquica cuya referencia se pasa en el parámetro lista.

Nota de compatibilidad: los parámetros apariencia e icono son obsoletos, siempre deben tomar el valor 0.

Nota: si desea personalizar el icono de cada elemento en la lista, utilice el comando **SET LIST ITEM PROPERTIES**.
Si no pasa el parámetro altoLinea, la altura de línea de una lista jerárquica es determinada por la fuente y el tamaño de fuente utilizado por el objeto. También puede pasar en parámetro altoLinea la altura de línea mínima de la lista jerárquica. Si el valor que pasa es superior a la altura de las líneas definida por la fuente y el tamaño de fuente utilizado, la altura de las líneas de la lista jerárquicas será el valor pasado.Pase 0 para definir la altura por defecto.

Nota: SET LIST PROPERTIES afecta la apariencia de los nodos en la lista jerárquica. Si prefiere personalizar el icono de cada elemento en la lista, utilice el comando **SET LIST ITEM PROPERTIES**.

El parámetro opcional dobleClic le permite definir que un doble clic en un elemento de la lista padre no provoque el despliegue o contracción de la sublista. Por defecto, un doble clic en un elemento de la lista padre provoca que su lista hijo se expanda o se contraiga. Sin embargo, algunas interfaces de usuario podrían necesita desactivar este mecanismo. Para hacer esto, pase 1 en el parámetro dobleClic.

Sólo se desactivará doble-clic. Los usuarios aún podrán expandir o contraer las sublistas haciendo clic en el nodo de la lista.

Si omite el parámetro dobleClic o pasa 0, se aplica el funcionamiento por defecto.

El parámetro opcional multiSeleccion le permite indicar si la lista debe aceptar selecciones múltiples.

Por defecto, como en versiones anteriores de 4D, usted no puede seleccionar varios elementos de una lista jerárquica simultáneamente. Si quiere que esta función esté disponible para la lista, pase el valor 1 en el parámetro multiSeleccion. En ese caso, las selecciones múltiples pueden efectuarse:

- manualmente, utilizando la combinación de teclas **Mayús**+**clic** para una selección continúa o **Ctrl**+**clic** (Windows) / **comando**+**clic** (Mac OS) para una selección discontinúa,
- por programación, utilizando los comandos **SELECT LIST ITEMS BY POSITION** y **SELECT LIST ITEMS BY REFERENCE**.

Si pasa 0 y omite el parámetro multiSeleccion, se aplicará el comportamiento por defecto.

El parámetro opcional editable le permite indicar si la lista debe ser editable por el usuario cuando se muestra como una lista de selección asociada a un campo o a una variable durante la entrada de datos. Cuando la lista es editable, un botón **Modificar** se añade en la ventana de la lista y el usuario puede añadir, borrar y ordenar los valores a través de un editor específico.
Si pasa 1 u omite el parámetro editable, la lista será editable; si pasa 0, no será editable.

Ejemplo

Desea rechazar la sublista de expandir/contraer al hacer doble clic. Puede escribir en el método de formulario:

```
Case of
   :(Form event=On Load)
      hlCities:=Load list("Cities") //cargar la lista de opciones Cities en el objeto de formulario hlCities
      SET LIST PROPERTIES(hlCities;0;0;0;1) //no permitir doble clic para expandir/contraer
End case
```

SORT LIST

SORT LIST (lista {; > ou <})

Parámetro	Tipo		Descripción
lista	ListRef	➡	Número de referencia de lista
> ou <	Operador	➡	Criterio de ordenación: > ordenar en orden ascendente, u < ordenar en orden descendente

Descripción

*El comando **SORT LIST** ordena la lista cuyo número de referencia se pasa en lista.*

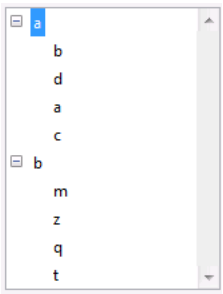
*Para ordenar en orden ascendente, pase >. Para ordenar en orden descendente, pase <. Si omite el parámetro de criterio de ordenación, **SORT LIST** ordena por defecto en orden ascendente.*

***SORT LIST** ordena todos los niveles de la lista; primero ordena los elementos de la lista, luego ordena los elementos en cada sublista (si hay), etc., a través de todos los niveles de la lista. Esta es la razón por la cual usted generalmente aplicará **SORT LIST** a una lista en un formulario. La ordenación de una sublista no es de mucho interés porque el orden cambiará por una llamada a un nivel superior.*

***SORT LIST** no cambia el estado actual de la lista y de las eventuales sublistas desplegado/contraído, ni del elemento actual. Sin embargo, como el elemento actual puede ser movido por la operación de ordenación, **Selected list items** podría devolver una posición diferente antes y después de la ordenación.*

Ejemplo

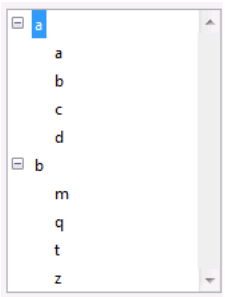
Dada la lista llamada hList, mostrada aquí en el entorno Aplicación:



Después de la ejecución de este código:

```
` Ordenar la lista y sublistas en orden ascendente
SORT LIST(hList;>)
```

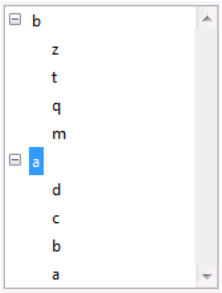
La lista se verá así:



Después de la ejecución de este código:

```
` Ordenar la lista y sus sublistas en orden decreciente
SORT LIST(hList;<)
REDRAW LIST(hList) ` NO olvide llamar REDRAW LIST de lo contrario la lista no se actualizará
```

La lista se ve de esta forma:



_o_REDRAW LIST

_o_REDRAW LIST (lista)

Parámetro	Tipo		Descripción
lista	ListRef	➡	Número de referencia de la lista

Nota de compatibilidad

Este comando es inútil a partir de la versión 11 de 4D. Todas las representaciones de listas jerárquicas ahora son redibujadas automáticamente.

Mensajes

- *ALERT*
- *CONFIRM*
- *DISPLAY NOTIFICATION*
- *GOTO XY*
- *MESSAGE*
- *MESSAGES OFF*
- *MESSAGES ON*
- *Request*

ALERT

ALERT (mensaje {; titulobotonOK})

Parámetro	Tipo		Descripción
mensaje	Cadena	➡	Mensaje a mostrar en la caja de diálogo de alerta
titulobotonOK	Cadena	➡	Título del botón OK

Descripción

*El comando **ALERT** muestra una caja de diálogo de alerta compuesta de un icono, de un mensaje y de un botón OK.*

Pase el mensaje a mostrar en el parámetro mensaje. Este mensaje puede tener hasta 255 caracteres. Si el mensaje no se ajusta al área del mensaje, puede aparecer truncado, dependiendo de su longitud y del ancho de los caracteres.

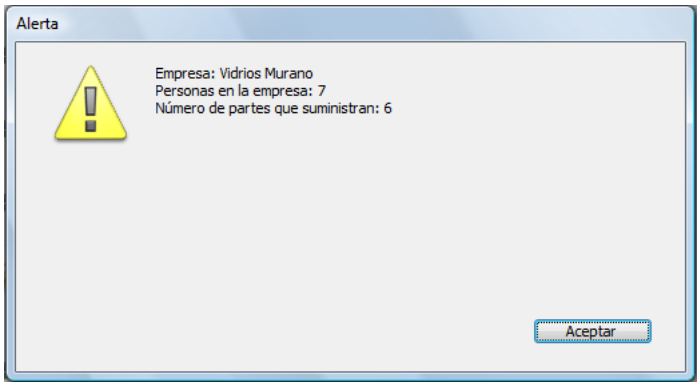
Por defecto, el título del botón OK es "Aceptar." Para cambiar el título del botón OK, pase el nuevo título en el parámetro opcional titulobotonOK. Si es necesario, el ancho del botón OK se redimensiona hacia la izquierda, de acuerdo al ancho del título personalizado que usted pasa.

Ejemplo 1

Este ejemplo muestra una caja de diálogo de alerta que muestra información sobre una empresa. Note que el mensaje contiene retornos de carro, que hacen que el texto pase a la línea siguiente:

```
ALERT("Empresa: "+[Empresas]Nombre+Char(13)+"Personas en la empresa: "+
String(Records in selection([Persona]))+Char(13)+"Número de partes que suministran: "+
String(Records in selection([Elementos])))
```

Esta línea de código muestra la siguiente caja de diálogo de alerta (en Windows):

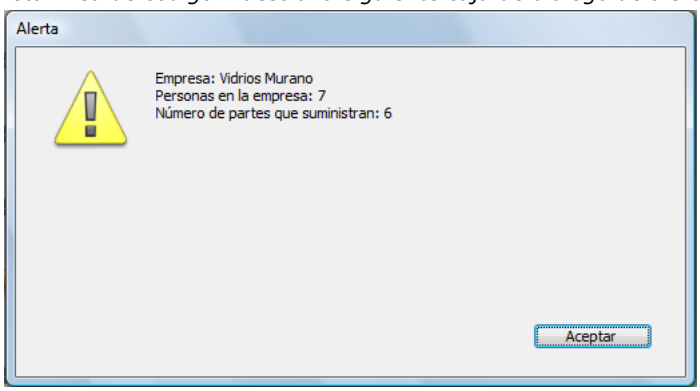


Ejemplo 2

Este ejemplo muestra una caja de diálogo de alerta que muestra la información de una empresa. Note que el mensaje contiene retornos de carro, haciendo que el texto pase a la siguiente línea:

```
ALERT("Empresa: "+[Empresas]Nombre+Char(13)+"Personas en la empresa: "+
String(Records in selection([Personas]))+Char(13)+"Número de partes que suministran: "+
String(Records in selection([Partes])))
```

Esta línea de código muestra la siguiente caja de diálogo de alerta (en Windows):

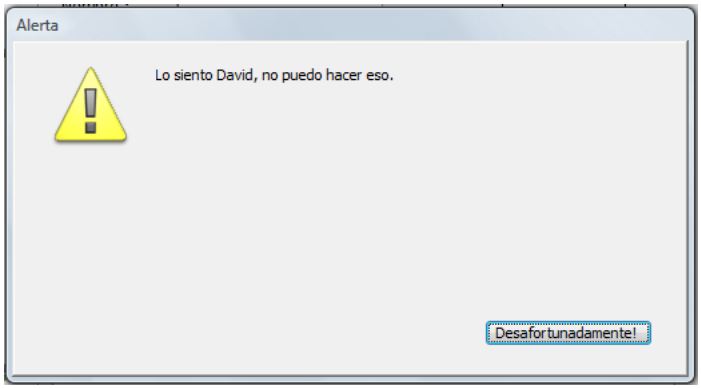


Ejemplo 3

La línea:

```
ALERT("Lo siento David, no puedo hacer eso";"Desafortunadamente!")
```

Muestra la siguiente caja de diálogo de alerta (en Windows):



CONFIRM

CONFIRM (mensaje {; titulobotonOK {; titulobotoncancel}})

Parámetro	Tipo		Descripción
mensaje	Cadena	➡	Mensaje a mostrar en la caja de diálogo de confirmación
titulobotonOK	Cadena	➡	Título del botón Aceptar
titulobotoncancel	Cadena	➡	Título del botón Cancelar

Descripción

El comando **CONFIRM** muestra una caja de diálogo de confirmación compuesta de un icono, un mensaje, un botón OK, y un botón Cancelar.

El mensaje a mostrar se pasa en el parámetro mensaje. Este mensaje puede tener hasta 255 caracteres. Si el mensaje no se ajusta al área de mensaje, puede aparecer truncado, dependiendo de su longitud y del ancho de los caracteres.

Por defecto, título del botón OK es "Aceptar" y el del botón Cancelar es "Cancelar." Para cambiar los títulos de estos botones, pase los nuevos títulos en los parámetros opcionales ok titulobotonOK y titulobotoncancel. Si es necesario, el ancho de los botones se redimensiona hacia la izquierda, de acuerdo al ancho de los títulos personalizados que usted pase.

El botón OK es el botón por defecto. Si el usuario hace clic en el botón OK o presiona Enter para aceptar la caja de diálogo, la variable sistema OK toma el valor 1. Si el usuario hace clic en el botón Cancel para cancelar la caja de diálogo, la variable sistema OK toma el valor 0.

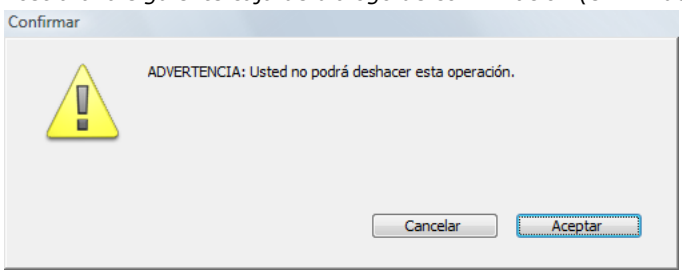
Consejo: no llame al comando **CONFIRM** desde un método de formulario o de objeto que maneje los eventos de formulario **On Activate** o **On Deactivate**; esto provocará un bucle infinito.

Ejemplo 1

La línea:

```
CONFIRM("ADVERTENCIA: Usted no podrá deshacer esta operación.")
If(OK=1)
   ALL RECORDS([Cosas Antiguas])
   DELETE SELECTION([Cosas Antiguas])
Else
   ALERT("Operación cancelada.")
End if
```

Mostrará la siguiente caja de diálogo de confirmación (en Windows):

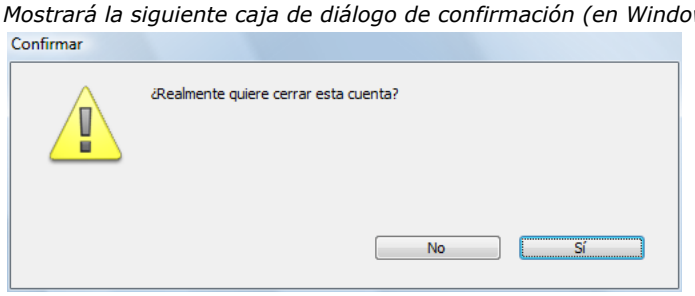


Ejemplo 2

La línea:

```
CONFIRM("¿Realmente quiere cerrar esta cuenta?";"Sí";"No")
```

Mostrará la siguiente caja de diálogo de confirmación (en Windows):



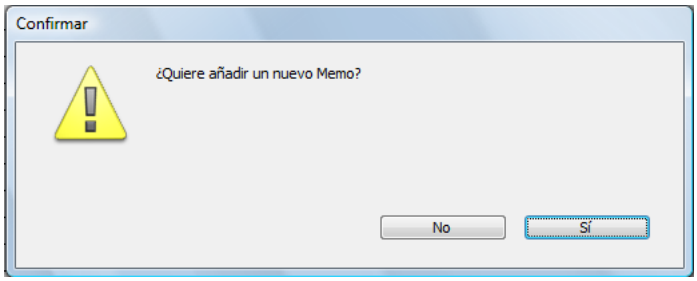
Ejemplo 3

Usted está desarrollando una aplicación 4D para el mercado internacional y escribió un método de proyecto que devuelve el texto traducido de la versión en inglés. Igualmente llena un array llamado <>asLocalizedUIMessages, donde almacena las palabras

más comunes. Al hacer esto, la línea:

```
CONFIRM(INTL Text("¿Quiere añadir un nuevo Memo?");<>asLocalizedUIMessages{kLoc_YES};
<>asLocalizedUIMessages{kLoc_NO})
```

Mostrará la siguiente caja de diálogo de confirmación:

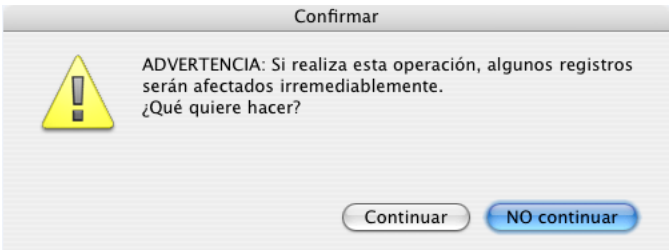


Ejemplo 4

La línea:

```
CONFIRM("ADVERTENCIA: Si realiza esta operación, algunos registros serán "+"afectados irremediablemente."+Char(13)+"¿Que quiere
hacer?";"NO continuar";"Continuar")
```

Mostrará la siguiente caja de diálogo de confirmación (en Macintosh):



DISPLAY NOTIFICATION

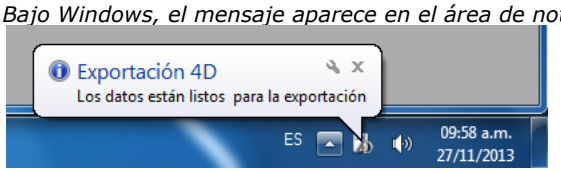
DISPLAY NOTIFICATION (titulo ; texto {; duracion})

Parámetro	Tipo		Descripción
titulo	Alpha	➡	Título de la notificación
texto	Alpha	➡	Texto de la notificación
duracion	Entero largo	➡	Duración de la visualización en segundos

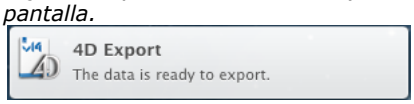
Descripción

*El comando **DISPLAY NOTIFICATION** muestra un mensaje de notificación al usuario:*
Este tipo de mensaje generalmente es utilizado por el sistema o por las aplicaciones para informar al usuario sobre un evento externo (desconexión de red, disponibilidad de una actualización, etc.).

- *Bajo Windows, el mensaje aparece en el área de notificación de la barra de tareas:*



- *Bajo OS X (versión 10.8 mínimo), el mensaje aparece en una pequeña ventana que en la esquina superior derecha de la pantalla.*



Note que conforme a las especificaciones de Apple, la notificación sólo se muestra cuando la aplicación no está en el primer plano. Sin embargo, el mensaje aún aparece en la lista del "notification center".

En titulo y texto, pase el título y el texto del mensaje a mostrar (en el ejemplo anterior, el título es "Exportación 4D"). Puede introducir hasta 255 caracteres.
Bajo Windows, se muestra la ventana del mensaje si no se detecta actividad en la máquina, o hasta que el usuario haga clic en la casilla de cerrar. El parámetro opcional duracion, modifica la duración de visualización por defecto. Note que la visualización de las notificaciones depende de las configuración del sistema.

Ejemplo

```
DISPLAY NOTIFICATION("4D Export";"The data is ready to export.")
```

GOTO XY

GOTO XY (x ; y)

Parámetro	Tipo		Descripción
x	Entero largo	➡	Posición x (horizontal) del cursor
y	Entero largo	➡	Posición y (vertical) del cursor

Descripción

*El comando **GOTO XY** se utiliza conjuntamente con el comando **MESSAGE** cuando usted muestra mensajes en una ventana abierta por el comando **Open window**.*

***GOTO XY** determina la posición del cursor de inserción de caracteres (un cursor invisible) para definir la ubicación del siguiente mensaje en la ventana.*

*La esquina superior izquierda representa las coordenadas 0,0. El cursor se ubica automáticamente en 0,0 cuando una ventana se abre y luego se ejecuta **ERASE WINDOW**.*

*Después de que **GOTO XY** defina la posición del cursor, puede utilizar **MESSAGE** para mostrar los caracteres en la ventana.*

Ejemplo 1

*Ver el ejemplo del comando **MESSAGE**.*

Ejemplo 2

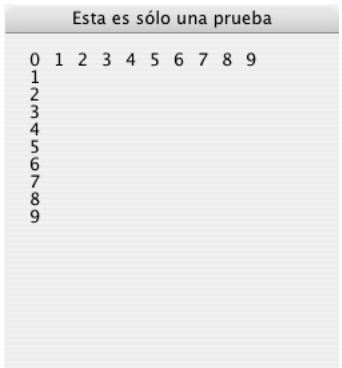
*Ver el ejemplo del comando **Milliseconds**.*

Ejemplo 3

El siguiente ejemplo:

```
Open window(50;50;300;300;5;"Esta es sólo una prueba")
For($vlFila;0;9)
   GOTO XY($vlFila;0)
   MESSAGE(String($vlFila))
End for
For($vlLinea;0;9)
   GOTO XY(0;$vlLinea)
   MESSAGE(String($vlLinea))
End for
$vhHoraInicio:=Current time
Repeat
Until((Current time-$vhHoraInicio)>?00:00:30?)
```

Muestra la siguiente ventana (en Macintosh) por 30 segundos:



MESSAGE

MESSAGE (mensaje)

Parámetro	Tipo		Descripción
mensaje	Cadena	➡	Mensaje a mostrar

Descripción

*El comando **MESSAGE** generalmente se utiliza para informar al usuario sobre alguna actividad. Muestra mensaje en la pantalla en una ventana de mensaje especial que se abre y cierra cada vez que usted llama a **MESSAGE**, a menos que usted trabaje con una ventana que abrió previamente utilizando **Open window** (ver los siguientes detalles). El mensaje es temporal y se borra tan pronto como se muestre un formulario o se detenga la ejecución del método. Si se ejecuta otro comando **MESSAGE**, el mensaje anterior se borra.*

*Si se abre una ventana con **Open window**, todas las llamadas posteriores al comando **MESSAGE** muestran los mensajes en esa ventana. La ventana se comporta como un terminal:*

- *Los mensajes sucesivos no borran los mensajes anteriores cuando se muestran en la ventana, se concadenan en mensajes existentes.*
- *Si un mensaje es más largo que la ventana, 4D inserta automáticamente un retorno a la línea.*
- *Si un mensaje tiene más líneas que la ventana, 4D automáticamente desplaza el mensaje en la ventana.*
- *Para controlar los saltos de línea automáticos, incluya retornos de carro, — **Char(13)** —, en su mensaje.*
- *Para mostrar el texto en un lugar en particular de la ventana, llame **GOTO XY**..*
- *Para borrar los contenidos de la ventana, llame **ERASE WINDOW**..*
- *La ventana es sólo una ventana de salida y su contenido no puede ser rediseñado cuando otras ventanas se colocan sobre ella.*
- *Puede modificar la fuente y el tamaño de los caracteres mostrados en la ventana a través de la página "Interfaz" en las Propiedades de la base.*

***Nota: MESSAGE** es compatible con el comando **Open form window**; sin embargo, en este contexto, el segundo parámetro * de **Open form window**, que guarda el tamaño y la posición de la ventana, no es compatible.*

Ejemplo 1

*El siguiente ejemplo procesa una selección de registros y llama **MESSAGE** para informar al usuario sobre el progreso de la operación:*

```
For($vlRegistro;1;Records in selection([todaTabla]))
   MESSAGE("Proceso del registro #"+String($vlRegistro))
   ` Hacer algo con el registro
   NEXT RECORD([todaTabla])
End for
```

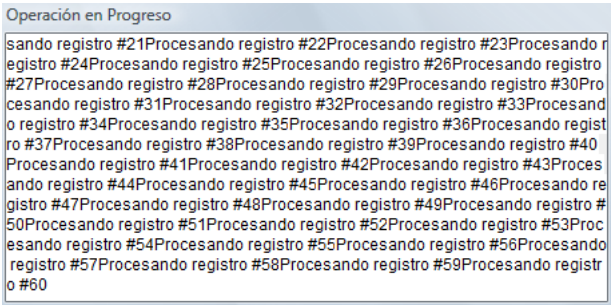
*La siguiente ventana aparece y desaparece cada vez que se llama **MESSAGE**:*

Ejemplo 2

*Con el fin de evitar la ventana "titilante", puede mostrar los mensajes en una ventana abierta utilizando **Open window**, como en este ejemplo:*

```
Open window(50;50;500;250;5;"Operación en progreso")
For($vlRegistro;1;Records in selection([todaTabla]))
   MESSAGE("Procesando registro #"+String($vlRegistro))
   ` Hacer algo con el registro
   NEXT RECORD([todaTabla])
End for
CLOSE WINDOW
```

El resultado es el siguiente (en Windows):

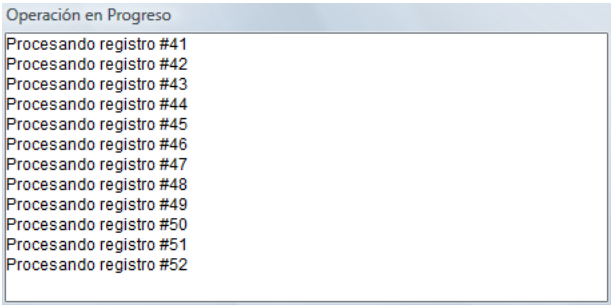


Ejemplo 3

Añadiendo un retorno de carro mejora la presentación:

```
Open window(50;50;500;250;5;"Operación en progreso")
For($vlRegistro;1;Records in selection([todaTabla]))
   MESSAGE("Procesando registro #"+String($vlRegistro)+Char(Carriage return))
  ` Hacer algo con el registro
   NEXT RECORD([todaTabla])
End for
CLOSE WINDOW
```

Este es el resultado (en Windows):

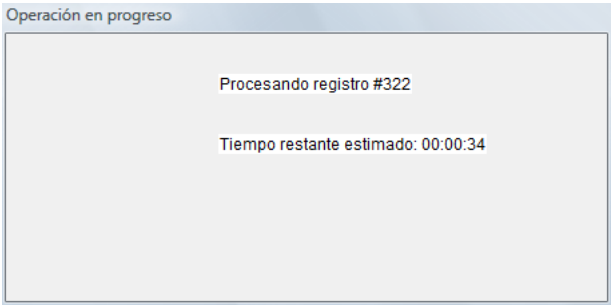


Ejemplo 4

*Utilizando **GOTO XY** y escribiendo algunas líneas adicionales:*

```
Open window(50;50;500;250;5;"Operación en progreso")
$vlNbRegistros:=Records in selection([todaTabla])
$vhStartTime:=Current time
For($vlRegistro;1;$vlNbRegistros)
   GOTO XY(5;2)
   MESSAGE("Procesando registro #"+String($vlRegistro)+Char(Carriage return))
  ` Hacer algo con el registro
   NEXT RECORD([todaTabla])
   GOTO XY(5;5)
   $vlResto:=(($vlNbRegistros/$vlRegistro)-1)*(Current time-$vhHoraInicio)
   MESSAGE("Tiempo restante estimado: "+Time string($vlResto))
End for
CLOSE WINDOW
```

El resultado es el siguiente (en Windows):



MESSAGES OFF

MESSAGES OFF
Este comando no requiere parámetros

Descripción

*Los comandos **MESSAGES OFF** y **MESSAGES ON** encienden y apagan los termómetros de progresión mostrados por 4D mientras se ejecutan operaciones de larga duración. Por defecto, se muestran los mensajes.*

Estas son las operaciones que pueden mostrar termómetros de progreso: aplicación de una fórmula, generación de un informe rápido, exportación de datos, importación de datos, ordenación, generación de un gráfico, búsqueda, búsqueda por formulario, búsqueda por fórmula.

La siguiente tabla lista los comandos que muestran termómetros de progreso:

APPLY TO SELECTION
Average
BUILD APPLICATION
DISTINCT VALUES
EXPORT DIF
EXPORT SYLK
EXPORT TEXT
_o_GRAPH TABLE
IMPORT DIF
IMPORT SYLK
IMPORT TEXT
Max
Min
ORDER BY
ORDER BY FORMULA
QR REPORT
QUERY
QUERY BY FORMULA
QUERY BY EXAMPLE
QUERY SELECTION
QUERY SELECTION BY FORMULA
REDUCE SELECTION
RELATE MANY SELECTION
RELATE ONE SELECTION
SCAN INDEX
Sum

***Nota para 4D Server:** a partir de 4D Server v14 R3, las ventanas de mensajes de progreso no se muestran en el servidor, estas operaciones se listan automáticamente en la **Ventana de administración de 4D Server** de la ventana de administración. Si desea forzar la visualización de estas ventanas de progreso, debe llamar al comando **MESSAGES ON** en el servidor.*

Ejemplo

El siguiente ejemplo suprime los termómetros de progreso antes de efectuar una ordenación y luego los restablece después de terminar la operación de ordenación:

```
MESSAGES OFF
ORDER BY([Direcciones];[Direcciones]ZIP;>;[Direcciones]Nombre2;>)
MESSAGES ON
```

MESSAGES ON

MESSAGES ON
Este comando no requiere parámetros

Descripción

*Ver la descripción del comando **MESSAGES OFF**.*

Request

Request (mensaje {; respuestaDefecto {; titulobotonOK {; titulobotoncancel}}}) -> Resultado

Parámetro	Tipo		Descripción
mensaje	Cadena	➡	Mensaje a mostrar en la caja de diálogo
respuestaDefecto	Cadena	➡	Valor por defecto en el área de entrada de texto
titulobotonOK	Cadena	➡	Título del botón Aceptar
titulobotoncancel	Cadena	➡	Título del botón Cancelar
Resultado	Cadena	⮌	Valor introducido por el usuario

Descripción

*El comando **Request** muestra una caja de diálogo compuesta de un mensaje, un área de entrada de texto, un botón **OK** y un botón **Cancelar**.*

El mensaje a mostrar se pasa en el parámetro mensaje. Si el mensaje no se ajusta al área de mensaje (por lo general alrededor de 50 caracteres, varía dependiendo del sistema y de la fuente utilizada), puede aparecer truncado.

*Por defecto, título del botón **OK** es "Aceptar" y el del botón **Cancelar** es "Cancelar." Para cambiar los títulos de estos botones, pase los nuevos títulos en los parámetros opcionales titulobotonOK y titulobotoncancel. Si es necesario, el ancho de los botones se redimensiona hacia la izquierda, de acuerdo al ancho de los títulos personalizados que usted pase.*

*El botón **OK** es el botón por defecto. Si el usuario hace clic en el botón **OK** o presiona **Intro** para aceptar la caja de diálogo, la variable sistema **OK** toma el valor 1. Si el usuario hace clic en el botón **Cancelar** para cancelar la caja de diálogo, la variable sistema OK toma el valor 0.*

*El usuario puede introducir texto en el área de entrada de texto. Para especificar un valor por defecto, pase el texto por defecto en el parámetro respuestaDefecto. Si el usuario hace clic en el botón **OK**, **Request** devuelve el texto. Si el usuario hace clic en **Cancelar**, **Request** devuelve una cadena vacía (""). Si la respuesta debe ser un valor numérico o una fecha, convierta la cadena devuelta por **Request** al tipo deseado con la ayuda de las funciones **Num** or **Date**.*

***Nota:** no llame el comando **Request** desde un método de formulario o de objeto que maneje los eventos de formulario On Activate o On Deactivate; esto provocará un bucle infinito.*

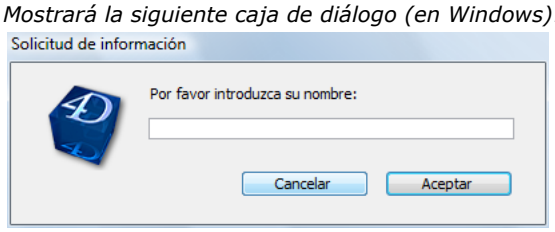
***Consejo:** si necesita obtener varias piezas de información del usuario, diseñe un formulario y preséntelo con **DIALOG**, en lugar de presentar una sucesión de cajas de diálogo de tipo **Request**.*

Ejemplo 1

La línea:

```
$vsPrompt:=Request("Por favor introduzca su nombre:")
```

Mostrará la siguiente caja de diálogo (en Windows):

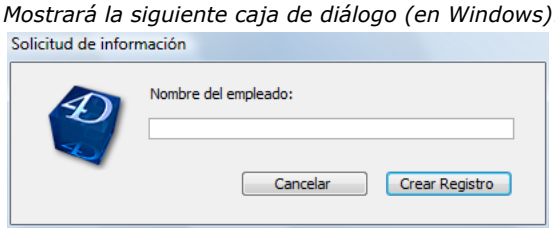


Ejemplo 2

La línea:

```
vsPrompt:=Request("Nombre del empleado:";"";"Crear Registro";"Cancelar")
If(OK=1)
   ADD RECORD([Empleados])
  ` Nota: vsPrompt luego se copia en el campo[Empleados]Apellido
  ` durante el evento On Load en el método de formulario
End if
```

Mostrará la siguiente caja de diálogo (en Windows):

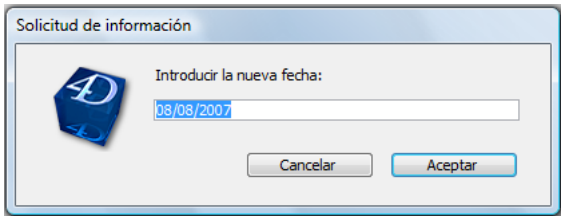


Ejemplo 3

La línea:

```
$vdPrompt:=Date(Request("Introduzca la nueva fecha:";String(Current date)))
```

Mostrará la siguiente caja de diálogo (en Windows):



Menús

- *Gestión de menús*
- *APPEND MENU ITEM*
- *Count menu items*
- *Count menus*
- *Create menu*
- *DELETE MENU ITEM*
- *DISABLE MENU ITEM*
- *Dynamic pop up menu*
- *ENABLE MENU ITEM*
- *Get menu bar reference*
- *Get menu item*
- *GET MENU ITEM ICON*
- *Get menu item key*
- *Get menu item mark*
- *Get menu item method*
- *Get menu item modifiers*
- *Get menu item parameter*
- *GET MENU ITEM PROPERTY*
- *Get menu item style*
- *GET MENU ITEMS*
- *Get menu title*
- *Get selected menu item parameter*
- *INSERT MENU ITEM*
- *Menu selected*
- *RELEASE MENU*
- *SET MENU BAR*
- *SET MENU ITEM*
- *SET MENU ITEM ICON*
- *SET MENU ITEM MARK*
- *SET MENU ITEM METHOD*
- *SET MENU ITEM PARAMETER*
- *SET MENU ITEM PROPERTY*
- *SET MENU ITEM SHORTCUT*
- *SET MENU ITEM STYLE*

Gestión de menús

Terminología:

*La documentación de los comandos de menús utiliza indiferentemente los términos **comando de menú y elemento de menú** cuando describe una línea de un menú.*

MenuRef y números de menús

*El lenguaje de 4D ofrece dos modos de manipulación de menús y barras de menús: por **referencias** o por **números**.*

- *La gestión de menús por **referencia** (MenuRef) es la nueva manera de administrar menús, introducida con la versión 11 de 4D. Este modo da acceso a funciones avanzadas tales como la creación de interfaces completamente dinámicas (menús creados "al vuelo" sin que deba existir necesariamente en el editor de menús) y la gestión de submenús jerárquicos multinivel.*
- *La gestión de menús y barras de menús por **número** está basada en los menús creados en el editor de menús en modo Diseño. Se asigna un número fijo a cada barra de menús y menú (correspondiente a su posición en el editor). Este número es utilizado por los comandos del lenguaje para designar la barra o el menú. El alcance de los comandos del lenguaje aplicados a los menús administrados por número es la barra de menús actual.Este comportamiento corresponde a las versiones anteriores de 4D y obedece a varias reglas (descritas a continuación en el párrafo "Gestión de menús por número"). Aún puede utilizarse pero no permite tomar ventaja de las nuevas funcionalidades ofrecidas a partir de la versión 11, particularmente la gestión dinámica de menús y la utilización de submenús jerárquicos: no es posible acceder a un submenú jerárquico utilizando un número.*

Los dos modos de gestión de menús son compatibles y pueden ser utilizados simultáneamente en sus interfaces. La mayoría de los comandos en el tema "Menús" aceptan indiferentemente números o referencias de menús.

*Sin embargo, la gestión de menús por referencia es recomendable ya que ofrece muchas más posibilidades. Observe que si su interfaz de menús está definida parcial o totalmente vía el editor de menús, es perfectamente posible trabajarla en forma de referencias utilizando los comandos **Get menu bar reference** y **GET MENU ITEMS**.*

Gestión de menús por referencia

Cuando los menús son manejados por medio de referencias MenuRef, no hay diferencia entre un menú y una barra de menús. En ambos casos, se trata de una lista de elementos. Sólo su uso es diferente. En el caso de una barra de menús, cada elemento corresponde a un menú, el cual está compuesto de elementos. Este también es el principio en el cual los menús jerárquicos están basados: cada elemento puede ser un menú y así sucesivamente.

Cuando un menú se administra por referencia, toda modificación efectuada sobre este menú durante la sesión pasa inmediatamente a cada instancia del menú y en todo proceso de la base.

MenuRef

Como las listas jerárquicas, todos los menús tienen una referencia única, gracias a la cual puede identificarse durante toda la sesión. Esta referencia, llamada por convención MenuRef, es un alfa numérico de 16 caracteres. Todos los comandos del tema "Menús" aceptan esta referencia, o un número de menú, para designar un menú o una barra de menús.

*Las referencias de menús pueden obtenerse utilizando los comandos **Create menu, Get menu bar reference** o **GET MENU ITEMS**.*

Gestión de menús por número

Barras de menús

Las barras de menús pueden definirse en el editor de menús en el entorno Diseño. Cuando son administradas por número, cada barra de menú es identificada por un número y por un nombre. La primera barra de menú (creada automáticamente por 4D) tiene el número 1 y se llama por defecto Barra de Menús #1. Puede renombrarla en el editor de menús. El nombre de una barra de menú puede contener hasta 31 caracteres y debe ser único.

*Barra de Menús #1 también es la barra de menús por defecto. Para abrir una aplicación con una barra de menús diferente de Barras de Menús #1, debe utilizar el comando **SET MENU BAR** en el **Método de base On Startup**.*

*No es posible modificar por programación el contenido mismo de una barra de menús, sin embargo los menús que la componen pueden ser modificados. El alcance de los comandos del lenguaje aplicados a menús estáticos es la barra de menús actual. En cada llamada al comando **SET MENU BAR** (sin el parámetro *), todos los menús y los comandos de menús vuelven a su estado original definido en el editor de menús.*

Cada barra de menús tiene por defecto tres menús: Archivo, Edición y Entorno.

• El menú Archivo tiene únicamente un comando de menú: Salir. La acción estándar Salir está asignada a él. Esta acción muestra una caja de diálogo de confirmación "¿Está seguro?" , si la caja de diálogo se valida sale de la aplicación 4D. De lo contrario, la operación se cancela.

***Nota:** en Mac OS X, el comando de menú creado asociado a la acción Salir se ubica automáticamente en el menú de la aplicación, cuando la base se ejecuta en este sistema.*

Puede renombrar el menú Archivo, añadirle comandos de menú o conservarlo tal cual. Se recomienda conservar siempre Salir como último comando del menú Archivo.

• El menú Edición contiene los comandos del menú edición estándar. Una acción estándar (Cancelar, Cortar, Copiar, etc.) se

asigna a cada comando de este menú. Puede añadir comandos a este menú o utilizar sus propios métodos de gestión de acciones de edición.

• El menú Entorno contiene el comando Volver al entorno Diseño. Este comando permite volver al modo Diseño (cuando está disponible) desde el modo Aplicación.

***Nota:** 4D administra automáticamente los menús sistema Ayuda, y aplicación (Mac OS X). Estos menús no pueden ser modificados, excepto por el comando Acerca de 4D, el cual puede administrarse utilizando el comando **SET ABOUT**.*

***Advertencia:** las barras de menú son "interproceso." Toda modificación efectuada en una barra de menú en el modo Diseño se reflejará en todos los procesos donde se utilice la barra de menús.*

Números de menús y de comandos de menú

*Como las barras de menú, los menús son numerados. El menú Archivo es generalmente el número 1. Los otros menús son numerados secuencialmente de izquierda a derecha (2, 3, 4, etc.). El menú Aplicación (Mac OS) está excluido de esta numeración. En ambas plataformas, el menú Ayuda también está excluido. Debe observarse que el comando **Count menus** no tiene en cuenta estos menús. Si, por ejemplo, su barra de menús está formada por los menús Archivo, Edición, Clientes, Facturas y Ayuda, **Count menus** devolverá 4 (ignorando los menús sistema mantenidos por 4D). La numeración de los menús es importante cuando está trabajando, por ejemplo, con la función **Menu selected**.*

Cuando un menú está asociado a un formulario, el esquema de numeración de menús es diferente. El primer menú añadido comienza con el número 2049. Para referenciar a un menú asociado, añada 2048 al número inicial del menú.

Los comandos de menú en cada menú se numeran secuencialmente de arriba a abajo, incluyendo los separadores. El comando de menú superior es el número 1.

Las barras de menús asociadas

Puede asociar una barra de menús a un formulario en las Propiedades del formulario (sección General). Este tipo de barra de menú se llama “barra de menús de formulario” en este documento.

Los menús de una barra de menús de formulario son añadidos a la barra de menús actual cuando el formulario se visualiza como un formulario de salida en el entorno Aplicación.
Las barras de menús de formularios son referenciadas por un número y un nombre. Si el número o el nombre de la barra de menús mostrada con el formulario actual es el mismo que el de la barra de menús asociada al formulario, ésta última no se muestra.

Por defecto, cuando un formulario se muestra con una barra de menús personalizada, los comandos de la barra de menús actual son desactivados, es decir que no tiene ningún efecto seleccionarlos. Puede modificar este funcionamiento seleccionando la opción Barra de menús activa en las Propiedades del formulario: en este caso, los comandos de la barra de menús actual permanecerán utilizables.

*En todos los casos, la selección de un comando de menú causa el envío de un evento **On Menu Selected** al método de formulario; puede utilizar el comando **Menu selected** para probar el menú seleccionado.*

Menús adjuntos

Los menús pueden estar adjuntos a las barras de menús. Si un menú adjunto se modifica utilizando uno de estos comandos, cada una de las instancias del menú reflejará los cambios. Para mayor información sobre menús adjuntos, consulte el Manual de Diseño.

Acciones estándar y métodos asociados con comandos de menús

Cada comando de menú puede tener un método de proyecto o una acción estándar asociada. Si no asigna un método o una acción estándar a un comando de menú, la selección de este comando provaoca la salida del modo Aplicación y el regreso al entorno Diseño. Si sólo está disponible el entorno Diseño o si el usuario no tiene acceso al entorno Diseño, esto provocará el cierre de la aplicación.

Las acciones estándar permiten efectuar varias operaciones asociadas a funciones sistema (copiar, salir, etc.) o de base de datos 4D (añadir registro, seleccionar todo, etc.).
Puede asignar a la vez una acción estándar y un método de proyecto a un comando de menú. En este caso, la acción estándar nunca se ejecuta; sin embargo 4D utiliza esta acción para activar/desactivar el comando de menú acorde al contexto actual y asociar una operación específica de acuerdo a la plataforma (por ejemplo, la acción Preferencias se pasa en el menú aplicación bajo Mac OS). Cuando un comando de menú está desactivado, el método de proyecto asociado no puede ejecutarse.

elementoMenu=-1

Con el fin de facilitar la manipulación de elementos de menú, 4D ofrece un atajo que puede utilizarse para especificar el último elemento añadido al menú: simplemente debe pasar -1 en el parámetro elementoMenu.

Este principio se puede utilizar en todos los comandos del tema "Menús" que trabajan con elementos de menús.

APPEND MENU ITEM

APPEND MENU ITEM (menu ; itemText {; subMenu {; proceso {; *}}})

Parámetro	Tipo		Descripción
menu	Entero largo	➡	Número de menú o referencia de menú
itemText	Texto	➡	Texto para los nuevos elementos de menú
subMenu	MenuRef	➡	Referencia del submenú asociado al elemento
proceso	Entero largo	➡	Número de referencia del proceso
*	Operador	➡	Si se pasa: considerar metacaracteres como caracteres estándar

Descripción

El comando **APPEND MENU ITEM** añade nueva líneas de menú al menú cuyo número o referencia se pasa en menu.

Si omite el parámetro proceso, **APPEND MENU ITEM** se aplica a la barra de menús del proceso actual. De lo contrario, **APPEND MENU ITEM** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.

Nota: si pasa un MenuRef en menu, el parámetro proceso es inútil y se ignorará.

Si no pasa el parámetro *, **APPEND MENU ITEM** le permite añadir una o varias líneas de menú en una sola llamada.

Las líneas a añadir se definen con el parámetro itemText de la siguiente manera:

- Separe cada elemento del siguiente con un punto y coma (;). Por ejemplo, "ItemText1;ItemText2;ItemText3".
- Para desactivar una línea: Coloque un paréntesis abierto (() en el texto de la línea.
- Para definir una línea de separación: Pase "-" o "(-" como texto de la línea.
- Para especificar un estilo de fuente para una línea: en el texto de la línea, ponga un signo menor que (<) seguido por uno de los siguientes caracteres:

<B Negrita
<I Cursiva
<U Subrayado

- Para añadir una marca de selección a una línea: en el texto de la línea, ponga un signo de admiración (!) seguido por el carácter que quiere como marca de selección. En Macintosh, el carácter se muestra; en Windows, se muestra una marca de selección sin importar que carácter pase.
- Para añadir un icono a una línea: en el texto de la línea, ponga un acento circunflejo (^) seguido por un carácter cuyo código más 208 es el número del recurso del icono Mac OS.
- Para añadir un atajo a una línea: en el texto de la línea, ponga una barra oblicua (/) seguida por el carácter de atajo para el elemento.
- (**A partir de 4D v16 R3**) si el elemento está asociado con una acción estándar, pase la constante ak standard action title en itemText para utilizar automáticamente el nombre de la acción localizada y la información de contexto (si la hay), por ejemplo "Undo <previous action>".

Nota: utilice menús que tengan un número razonable de líneas. Por ejemplo, si quiere mostrar más de 50 elementos, considere utilizar un área de desplazamiento en un formulario en lugar de un menú.

Si pasa el parámetro *, los caracteres "especiales" (; (!...) incluidos en el texto del elemento se considerarán como caracteres estándar y no como metacaracteres. Esto permite crear elementos de menú tales como "**Copiar (especial)...**" o "**Buscar/Reemplazar...**". Note que cuando se pasa el parámetro *, no puede crear varios elementos en una sola llamada ya que el carácter ";" es considerado como un carácter estándar.

Nota: los comandos **GET MENU ITEMS** y **Get menu item** devolverán o no los metacaracteres en el texto de un elemento de menú dependiendo de cómo fue creado: si fue creado con la opción *, los metacaracteres serán devueltos como caracteres estándar.

El parámetro opcional subMenu permite designar un menú como elemento añadido y por lo tanto definir un submenú jerárquico. Debe pasar en este parámetro una referencia de menú (cadena de tipo MenuRef) especificando un menú creado por ejemplo, utilizando el comando **Create menu**. Si el comando añade más de un elemento de menú, el submenú se asocia a la primera línea.

Importante: las nuevas líneas no tienen métodos o acciones asociados. Estos deben asociarse con los elementos utilizando los comandos **SET MENU ITEM PROPERTY** o **SET MENU ITEM METHOD**, o los elementos también pueden ser administradas por un método de formulario utilizando el comando **Menu selected**.

Ejemplo

Este ejemplo añade los nombres de las fuentes disponibles en un menú Fuentes, el cual en este ejemplo es el sexto menú de la barra de menús actual:

```
` En el método base On Startup
` La lista de fuentes se carga y el texto de la línea de menú se construye
FONT LIST(◇asFuenteDisp)
◇atFuenteMenuItems:=""
For($vlFont;1;Size of array(◇asFuenteDisp))
   ◇atFuenteMenuItems:=◇atFuenteMenuItems+";"+◇asFuenteDisp{$vlFont}
End for
```

Entonces, en todo método de formulario o de proyecto, puede escribir:

```
APPEND MENU ITEM(6;◇atFuenteMenuItems)
```

Count menu items

Count menu items (menu {; proceso}) -> Resultado

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	→	Número de menú
proceso	Entero largo	→	Número de referencia del proceso
Resultado	Entero largo	↩	Número de líneas de menú en el menú

Descripción

El comando ***Count menu items*** *devuelve el número de los elementos del menú presentes en el menú cuyo número o referencia se pasa en menu.*

Si omite el parámetro proceso, ***Count menu items*** *se aplica a la barra de menús del proceso actual. De lo contrario,* ***Count menu items*** *se aplica a la barra de menús del proceso cuyo número de referencia se pasa en proceso.*

Nota*: si pasa un parámetro MenuRef en menu, el parámetro proceso es inútil y se ignorará.*

Count menus

Count menus {(proceso)} -> Resultado

Parámetro	*Tipo*		*Descripción*
proceso	*Entero largo*	→	*Número de referencia de proceso*
Resultado	*Entero largo*	↩	*Número de menús de la barra de menús actual*

Descripción

*El comando **Count menus** devuelve el número de menús presentes en la barra de menús.*

*Si omite el parámetro proceso, **Count menus** se aplica a la barra de menús del proceso actual. De lo contrario, **Count menus** se aplica a la barra de menús del proceso cuyo número de referencia se pasa en proceso.*

Create menu

Create menu {(menu)} -> Resultado

Parámetro	Tipo		Descripción
menu	MenuRef, Entero largo, Cadena	➡	Referencia del menú o Número o nombre de barra de menús
Resultado	MenuRef	⮌	Referencia del menú

Descripción

*El comando **Create menu** permite crear un nuevo menú en memoria. Este menú sólo existirá en memoria y no se añadirá al editor de menú en el entorno Diseño. Toda modificación efectuada a este menú durante la sesión repercutirá inmediatamente en todas las instancias de este menú y en todos los procesos de la base.*

El comando devuelve un identificador único de tipo MenuRef para el nuevo menú.

- *Si no pasa el parámetro opcional menu, el menú será creado vacío. Debe construirlo y administrarlo utilizando los comandos **RELEASE MENU**, **SET MENU ITEM**, etc.*
- *Si pasa el parámetro menu, el menú creado será una copia exacta del menú fuente designado por este parámetro. Todas las propiedades del menú fuente, incluyendo los submenús asociados, serán aplicadas al nuevo menú. Note que una nueva referencia MenuRef se crea para el menú fuente y para cada submenú asociado existente.*

En el parámetro menu, puede pasar una referencia de menú válida, o el número o nombre de una barra de menús definida en el entorno Diseño. En este último caso, el nuevo menú estará constituido por los menús y submenús de la barra de menús fuente.

***Nota:** si pasa un valor invalido en menu, se crea un menú vacío.*

*Un menú creado por este comando puede ser utilizado como barra de menús utilizando el comando **SET MENU BAR**.*

*Cuando ya no necesite el menú creado por **Create menu**, recuerde llamar al comando **RELEASE MENU** para liberar la memoria que está siendo utilizada.*

Ejemplo

*Consulte el ejemplo del comando **SET MENU BAR**.*

DELETE MENU ITEM

DELETE MENU ITEM (menu ; lineamenu {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Número de menú o Referencia de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 por el último elemento añadido
proceso	Entero largo	➡	Número de referencia de proceso

Descripción

*El comando **DELETE MENU ITEM** elimina la línea de menú cuyo número o referencia de menú y número de elemento usted pasó en menu y menuItem. Puede pasar -1 en menuItem para indicar el último elemento añadido a menu.*

*Si la línea de menú especificada menu y menuItem es un menú administrado por referencia y creado, por ejemplo, utilizando el comando **Create menu**, **DELETE MENU ITEM** borrará únicamente la instancia de menuItem en menu. El submenú referenciado por menuItem continuará existiendo en memoria. Debe utilizar el comando **RELEASE MENU** para borrar definitivamente un menú que es administrado por referencia.*

*Este comando también funciona con una barra de menús creada con el comando **Create menu** e instalada con el comando **SET MENU BAR**.*

*Si omite el parámetro proceso, **DELETE MENU ITEM** se aplica a la barra de menús del proceso actual. De lo contrario, **DELETE MENU ITEM** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

***Nota:** si pasa un MenuRef en menu, el parámetro proceso es inútil y será ignorado.*

***Nota:** por consistencia en la interfaz del usuario, no se conserva un menú que no tenga líneas.*

DISABLE MENU ITEM

DISABLE MENU ITEM (menu ; lineamenu {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Número de menú o Referencia del menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 por el último elemento añadido
proceso	Entero largo	➡	Número de referencia del proceso

Descripción

El comando ***DISABLE MENU ITEM*** *desactiva el comando de menú cuyo número o referencia de menú y número de elemento usted pasó en menu y menuItem. Puede pasar -1 en menuItem para indicar el último elemento añadido a menu.*

Si omite el parámetro proceso, ***DISABLE MENU ITEM*** *se aplica a la barra de menús del proceso actual. De lo contrario,* ***DISABLE MENU ITEM*** *se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

Si el parámetro menuItem designa un submenú jerárquico, todos los elementos de este menú y de los eventuales submenús son desactivados. Este comando también funciona con una barra de menús creada con el comando ***Create menu*** *e instalada con el comando* ***SET MENU BAR.***

Nota: *si pasa un MenuRef en menu, el parámetro proceso es inútil y será ignorado.*

Tip: *para activar/desactivar todas las líneas de menús de una vez, pase 0 (cero) en menuItem.*

Dynamic pop up menu

Dynamic pop up menu (menu {; pordefecto {; CoordX ; CoordY}}) -> Resultado

Parámetro	Tipo		Descripción
menu	MenuRef	➡	Referencia de menú
pordefecto	Cadena	➡	Parámetro del elemento seleccionado por defecto
CoordX	Entero largo	➡	Coordenada X de la esquina superior izquierda
CoordY	Entero largo	➡	Coordenada Y de la esquina superior izquierda
Resultado	Cadena	↩	Parámetro del elemento de menú seleccionado

Descripción

*El comando **Dynamic pop up menu** hace aparecer un menú desplegable jerárquico en la ubicación actual del ratón o en la ubicación definida por los parámetros opcionales Coordx y Coordy.*

*El menú jerárquico utilizado debe haber sido creado con el comando **Create menu**. La referencia devuelta por **Create menu** debe pasarse en el parámetro menu.*

***Nota:** el comando **Pop up menu** (tema "Interfaz de usuario") puede utilizarse parar crear menús pop-up menus basados en texto.*

Conforme con las reglas estándar de la interfaz, este comando generalmente debe ser llamado en respuesta a un clic derecho, o cuando el botón se mantiene presionado por un cierto periodo de tiempo (menú contextual por ejemplo).

*El parámetro opcional pordefecto puede utilizarse para definir un elemento del menú desplegable seleccionado por defecto cuando aparece el menú. En este parámetro, pase una cadena personalizada asociada a la referencia del elemento de menú. Esta cadena debe haber sido definida de antemano con la ayuda del comando **SET MENU ITEM PARAMETER**. Si no pasa este parámetro, el primer elemento del menú será seleccionado por defecto.*

Los parámetros opcionales CoordX y CoordY pueden ser utilizados para especificar la ubicación del menú desplegable a mostrar. En los parámetros CoordX y CoordY, pase las coordenadas horizontal y vertical respectivamente, de la esquina superior izquierda del menú. Estas coordenadas deben ser expresadas en píxeles en el sistema de coordenadas local del formulario actual. Estos dos parámetros deben pasarse juntos; si se pasa sólo uno de ellos, el otro será ignorado.

Si quiere mostrar un menú desplegable asociado a un botón 3D, no pase los parámetros opcionales CoordX y CoordY. En este caso, 4D calcula automáticamente la ubicación del menú respecto al botón en función de los estándares de interfaz de la plataforma actual (el botón 3D debe tener la propiedad "Con menú pop-up/relacionado" o "Con menú pop-up/Separado").

*Si un elemento de menú ha sido seleccionado, el comando devuelve su cadena de caracteres personalizada asociada (tal como se ha definido utilizando el comando **SET MENU ITEM PARAMETER**). De lo contrario, el comando devuelve una cadena vacía.*

***A partir de 4D v16 R3:** si una acción estándar está asociada a un elemento de menú, es tenida en cuenta por el comando **Dynamic pop up menu** en varios niveles:*

- *Si una acción estándar asociada no está activada (es decir, no se puede invocar) en el contexto del menú emergente, el elemento se oculta automáticamente. Puede saber si una acción se activa utilizando el comando **Get action info**.*
- *Los elementos con una acción asociada se seleccionan automáticamente, sin marcar o "mezclados" según la selección.*
- *Si el título de la acción se ha ajustado al elemento usando la constante ak standard action title, el nombre localizado se mostrará en el menú.*
- *Cuando se selecciona el elemento, se invoca la acción estándar asociada (la ejecución es asíncrona).*

Ejemplo

Este código permite crear un menú emergente dinámico jerárquico basado en acciones estándar:

```
C_TEXT($refMainContextMenu;$refMenuEdit)
$refMainContextMenu:=Create menu
APPEND MENU ITEM($refMainContextMenu;"-")
APPEND MENU ITEM($refMainContextMenu;ak standard action title)
SET MENU ITEM PROPERTY($refMainContextMenu;-1;Associated standard action;ak select all)
APPEND MENU ITEM($refMainContextMenu;ak standard action title)
SET MENU ITEM PROPERTY($refMainContextMenu;-1;Associated standard action;ak clear)
APPEND MENU ITEM($refMainContextMenu;ak standard action title)
SET MENU ITEM PROPERTY($refMainContextMenu;-1;Associated standard action;ak copy)
APPEND MENU ITEM($refMainContextMenu;ak standard action title)
SET MENU ITEM PROPERTY($refMainContextMenu;-1;Associated standard action;ak cut)
APPEND MENU ITEM($refMainContextMenu;ak standard action title)
SET MENU ITEM PROPERTY($refMainContextMenu;-1;Associated standard action;ak paste)
APPEND MENU ITEM($refMainContextMenu;"-")
 //sub menu text edit
$refMenuEdit:=Create menu
APPEND MENU ITEM($refMenuEdit;ak standard action title)
SET MENU ITEM PROPERTY($refMenuEdit;-1;Associated standard action;ak font bold)
SET MENU ITEM SHORTCUT($refMenuEdit;-1;Character code("B"))
APPEND MENU ITEM($refMenuEdit;ak standard action title)
SET MENU ITEM PROPERTY($refMenuEdit;-1;Associated standard action;ak font italic)
SET MENU ITEM SHORTCUT($refMenuEdit;-1;Character code("I"))
```

```
APPEND MENU ITEM($refMenuEdit;ak standard action title)
SET MENU ITEM PROPERTY($refMenuEdit;-1;Associated standard action;ak font linethrough)
SET MENU ITEM SHORTCUT($refMenuEdit;-1;Character code("L"))
APPEND MENU ITEM($refMenuEdit;ak standard action title)
SET MENU ITEM PROPERTY($refMenuEdit;-1;Associated standard action;ak font underline)
SET MENU ITEM SHORTCUT($refMenuEdit;-1;Character code("U"))
APPEND MENU ITEM($refMenuEdit;ak standard action title)
SET MENU ITEM PROPERTY($refMenuEdit;-1;Associated standard action;ak font show dialog)
APPEND MENU ITEM($refMainContextMenu;"Edit";$refMenuEdit)

paramRef:=Dynamic pop up menu($refMainContextMenu)
```

ENABLE MENU ITEM

ENABLE MENU ITEM (menu ; lineamenu {; proceso})

Parámetro	*Tipo*		*Descripción*
menu	*Entero largo, MenuRef*	➡	*Número de menú o Referencia del menú*
lineamenu	*Entero largo*	➡	*Número de línea de menú o -1 por el último elemento añadido*
proceso	*Entero largo*	➡	*Número de referencia del proceso*

Descripción

El comando ***ENABLE MENU ITEM*** *activa el comando de menú cuyo número o referencia de menú y número de elemento usted pasó en menu y menuItem. Puede pasar -1 en menuItem para indicar el último elemento añadido a menu.*

Si omite el parámetro proceso, ***ENABLE MENU ITEM*** *se aplica a la barra de menús del proceso actual. De lo contrario,* ***ENABLE MENU ITEM*** *se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

Nota: *si pasa un MenuRef en menu, el parámetro proceso es inútil y será ignorado.*

Tip: *para activar/desactivar todas los comandos de menús de una vez, pase 0 (cero) en menuItem.*

Get menu bar reference

Get menu bar reference {(proceso)} -> Resultado

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número de referencia del proceso
Resultado	MenuRef	⮌	Identificador de la barra de menús

Descripción

El comando ***Get menu bar reference*** *devuelve la identificación única de la barra de menús actual o de la barra de menús de un proceso específico.*

Si la barra de menús fue creada por el comando ***Create menu****, este identificación corresponde a la referencia única del menú creado. De lo contrario, el comando devuelve una identificación(*). interna específica. En todos los casos, esta identificación, MenuRef puede utilizarse para referenciar la barra de menús par todos los otros comandos del tema.*

(*) En las versiones 64 bits de 4D, este ID específico es temporal y deja de ser válido tan pronto como se llame a otra barra de menú con ***SET MENU BAR****. Si desea conservar la referencia de un menú creado en el editor de menú, debe copiarlo en la memoria utilizando* ***Create menu****. Por ejemplo:*

```
$vEditorRef:=Get menu bar reference(Frontmost process) //menú creado en el editor de barras de menú
$vMenuRef:=Create menu($vEditorRef) //copia el menú
SET MENU BAR(2) //instala otra barra de menú
... // ejecutar código
SET MENU BAR($vMenuRef) //regresa a la barra de menú inicial
```

El parámetro proceso puede utilizarse para designar el proceso del que quiere obtener la identificación de la barra de menús actual. Si omite este parámetro, el comando devuelve la identificación de la barra de menús del proceso actual.

Ejemplo

Consulte el ejemplo del comando ***GET MENU ITEMS****.*

Get menu item

Get menu item (menu ; lineamenu {; proceso}) -> Resultado

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Número de menú o Referencia de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 por el último elemento añadido
proceso	Entero largo	➡	Número de referencia del proceso
Resultado	Cadena	⮌	Texto del elemento de menú

Descripción

El comando ***Get menu item*** *devuelve el texto del comando de menú cuyos números de menú y de comando se pasan en menu y menuItem. Puede pasar -1 en menuItem para indicar el último elemento añadido a menu.*

Si omite el parámetro proceso, ***Get menu item*** *se aplica a la barra de menús del proceso actual. De lo contrario,* ***Get menu item*** *se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

Nota: *si pasa un MenuRef en menu, el parámetro proceso es inútil y se ignorará.*

GET MENU ITEM ICON

GET MENU ITEM ICON (menu ; lineamenu ; refIcono {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Referencia de menú o número de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 para el último elemento añadido al menú
refIcono	Variable texto, Variable entero largo	⬅	Nombre o número de imagen asociado con la línea de menú
proceso	Entero largo	➡	Número de proceso

Descripción

*El comando **GET MENU ITEM ICON** devuelve, en la variable refIcono, la referencia del icono asociado a la línea de menú designada por los parámetros menu y menuItem. Esta referencia es el nombre o número de la imagen en la librería de imágenes.*

Puede pasar -1 en lineaMenu para especificar la última línea añadida a menu.

En menu, puede pasar una referencia de menú (MenuRef) o un número de menú. Si pasa una referencia de menú, el parámetro proceso no será necesario y si se pasa seignorará. Si pasa un número de menú, el comando tendrá en cuenta el menú correspondiente en la barra de menús principal del proceso actual. Si quiere designar otro proceso, pase su número en el parámetro opcional proceso.

Si el icono se ha especificado utilizando la librería de imágenes el comando devuelve o bien el nombre o el número de la imagen dependiendo del tipo de variable pasada en este parámetro. Si el icono se ha especificado utilizando una imagen almacenada en la carpeta Resources de la base, el comando devuelve el nombre de la imagen en refIcono.

Si no atribuye un tipo específico a la variable refIcono, por defecto, se devuelve el nombre de la imagen (tipo texto).

Si ningún icono está asociado a la línea de menú, el comando devuelve una imagen vacía.

Get menu item key

Get menu item key (menu ; lineamenu {; proceso}) -> Resultado

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Número de menú o Referencia de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 por el último elemento añadido
proceso	Entero largo	➡	Número de referencia de proceso
Resultado	Entero largo	⮌	Código de caracter de la tecla de atajo estándar asociada a la línea de menú

Descripción

*El comando **Get menu item key** devuelve el código del atajo **Ctrl** (Windows) o **Comando** (Mac OS) para el comando de menú cuyo número o referencia de menú se pasa en menu y cuyo número de comando se pasa en menuItem. Puede pasar -1 en menuItem para indicar el último elemento añadido a menu.*

*Si omite el parámetro proceso, **Get menu item key** se aplica a la barra de menús del proceso actual. De lo contrario, **Get menu item key** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

***Nota:** si pasa un MenuRef en menu, el parámetro proceso es inútil y se ignorará.*

*Si el comando de menú no tiene ningún atajo asociado o si el parámetro menuItem designa un submenú jerárquico, **Get menu item key** devuelve 0 (cero).*

Ejemplo

Para obtener un atajo asociado con un comando de menú, es útil implementar una estructura de programación del siguiente tipo:

```
If(Get menu item key(mimenu;1)#0)
   $modificadores:=Get menu item modifiers(mymenu;1)
   Case of
      :($modificadores=Option key mask)
         ...
      :($modificadores=Shift key mask)
         ...
      :($modificadores=Option key mask+Shift key mask)
         ...
   End case
End if
```

Get menu item mark

Get menu item mark (menu ; lineamenu {; proceso}) -> Resultado

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Número de menú o Referencia de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 por el último elemento añadido
proceso	Entero largo	➡	Número de referencia de proceso
Resultado	Cadena	↩	Marca de línea del menú actual

Descripción

*El comando **Get menu item mark** devuelve la marca de la línea de menú cuyo número o referencia de menú y número de línea se pasan en menu y menuItem. Puede pasar -1 en menuItem para indicar el último elemento añadido a menu.*

*Si omite el parámetro proceso, **Get menu item mark** se aplica a la barra de menús del proceso actual. De lo contrario, **Get menu item mark** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

***Nota:** si pasa un MenuRef en menu, el parámetro proceso es inútil y será ignorado.*

*Si la línea de menú no tiene marca o si el parámetro menuItem especifica un submenú jerárquico, **Get menu item mark** devuelve una cadena vacía.*

***Nota:** para mayor información sobre las marcas de las líneas de menús en Macintosh y Windows, consulte la descripción del comando **SET MENU ITEM MARK**.*

Ejemplo

El siguiente ejemplo invierte la marca de una línea de menú:

```
SET MENU ITEM MARK($vlMenu;$vlItem;Char(18)*Num(Character code(Get menu item mark($vlMenu;$vlItem))#18))
```

Get menu item method

Get menu item method (menu ; lineaMenu {; proceso}) -> Resultado

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Referencia de menú o Número de menú
lineaMenu	Entero largo	➡	Número de línea de menú o -1 para el último elemento añadido al menú
proceso	Entero largo	➡	Número de proceso
Resultado	Cadena	➲	Nombre del método

Descripción

*El comando **Get menu item method** devuelve el nombre del método de proyecto 4D asociado a la línea de menú designada por los parámetros menu y lineaMenu.*

Puede pasar -1 en lineaMenu con el fin de especificar el último elemento añadido al menu.

En menu, puede pasar una referencia de menú (MenuRef) o un número de menú. Si pasa una referencia de menú, el parámetro proceso no será necesario y si se pasa será ignorado. Si pasa un número de menú, el comando tomará el menú correspondiente en la barra de menús principal del proceso actual. Si quiere designar otro proceso, pase su número en el parámetro opcional proceso.

El comando devuelve el nombre del método 4D como una cadena de caracteres (expresión). Si ningún método está asociado a la línea de menú, el comando devuelve una cadena vacía.

Get menu item modifiers

Get menu item modifiers (menu ; lineaMenu {; proceso}) -> Resultado

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Referencia de menú o número de menú
lineaMenu	Entero largo	➡	Número de línea de menú o -1 para el último elemento añadido al menú
proceso	Entero largo	➡	Número de proceso
Resultado	Entero largo	↩	Tecla(s) de modificación asociada(s) a la línea de menú

Descripción

*El comando **Get menu item modifiers** devuelve los modificadores adicionales asociados a los atajos de teclado estándar de la línea de menú designada por los parámetros menu y lineaMenu.*

*El atajo estándar está compuesto por la tecla **Comando** (Mac OS) o **Ctrl** (Windows) y de una tecla personalizada. El atajo estándar se administra utilizando los comandos **SET MENU ITEM SHORTCUT** y **Get menu item key**.*

*Los modificadores adicionales son la tecla **Mayús** y la tecla Opción (Mac OS) /Alt (Windows). Estos modificadores sólo pueden utilizarse cuando un atajo estándar ha sido definido de antemano.*

El valor del número devuelto por el comando corresponde al código de las teclas de modificación adicionales. Los códigos de las teclas son los siguientes:

- ***Mayús** = 512*
- ***Opción** (Mac OS) o **Alt** (Windows) = 2048*

Si se utilizan ambas teclas, sus valores se combinan.

***Nota:** puede evaluar el valor devuelto utilizando las constantes Shift key mask y Option key mask del tema "".*

Si la línea de menú no tiene una tecla de modificación asociada, el comando devuelve 0.

Puede pasar -1 en lineaMenu con el fin de especificar el último elemento añadido a menu.

En menu, puede pasar una referencia de menú (MenuRef) o un número.

Si pasa una referencia de menú, el parámetro proceso es inútil y será ignorado si se pasa.

Si pasa un número de menú, el comando tomará el menú correspondiente en la barra de menús principal del proceso actual. Si quiere designar otro proceso, pase su número en el parámetro opcional proceso.

Ejemplo

*Consulte el ejemplo del comando **Get menu item key**.*

Get menu item parameter

Get menu item parameter (menu ; lineaMenu) -> Resultado

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	→	Referencia de menú o número de menú
lineaMenu	Entero largo	→	Número de línea de menú o -1 para la última línea añadida al menú
Resultado	Cadena	←	Parámetro personalizado de la línea de menú

Descripción

El comando ***Get menu item parameter*** *devuelve la cadena de caracteres personalizada asociada a la línea de menú designada por los parámetros menu y lineaMenu. Esta cadena debe haber sido definida previamente utilizando el comando* ***SET MENU ITEM PARAMETER****.*

GET MENU ITEM PROPERTY

GET MENU ITEM PROPERTY (menu ; lineaMenu ; propiedad ; valor {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo	➡	Referencia de menú o número de menú
lineaMenu	Entero largo	➡	Número de línea de menú o -1 para el último elemento añadido al menú
propiedad	Cadena	➡	Tipo de propiedad
valor	Expresión	⬅	Valor de la propiedad
proceso	Entero largo	➡	Número del proceso

Descripción

*El comando **GET MENU ITEM PROPERTY** devuelve, en el parámetro valor, el valor actual de la línea de menú designada por los parámetros menu y lineaMenu.*
Puede pasar -1 en lineaMenu para especificar el último elemento añadido a menu.

En menu, puede pasar una referencia de menú (MenuRef) o un número de menú. Si pasa una referencia de menú, el parámetro proceso no es necesario y será ignorado si se pasa. Si pasa un número de menú, el comando tendrá en cuenta el menú correspondiente en la barra de menús principal del proceso actual. Si quiere designar otro proceso, pase su número en el parámetro opcional proceso.

*En el parámetro propiedad, pase la propiedad para la cual quiere obtener el valor. Puede utilizar una de las constantes del tema "**Propiedades de ítem de menú**" o una cadena correspondiente a una propiedad personalizada. Para mayor información sobre las propiedades de los menús y sus valores, consulte la descripción del comando **SET MENU ITEM PROPERTY**.*

***Nota de compatibilidad**: por defecto, si la variable valor no se escribe o declara explícitamente como texto, el comando devolverá un nombre **Acción estándar**. Si desea obtener un valor numérico tal como se define en el tema de la constante **Valores para acción estándar asociada** (obsoleto), debe declarar la variable valor como entero largo.*

Get menu item style

Get menu item style (menu ; lineamenu {; proceso}) -> Resultado

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Número de menú o Referencia de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 para el último elemento añadido
proceso	Entero largo	➡	Process reference number
Resultado	Entero largo	⮌	Estilo del comando de menú

Descripción

*El comando **Get menu item style** devuelve el estilo de fuente de la línea de menú cuyo número o referencia se pasa en menu y cuyo número de elemento se pasa en menuItem. Puede pasar -1 en menuItem para indicar el último elemento añadido a menu.*

*Si omite el parámetro proceso, **Get menu item style** se aplica a la barra de menús del proceso actual. De lo contrario, **Get menu item style** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

***Nota:** si pasa un MenuRef en menu, el parámetro proceso es inútil y se ignorará.*

***Get menu item style** devuelve una combinación (uno o una suma) de las siguientes constantes predefinidas:*

Constante	Tipo	Valor
Plain	Entero largo	0
Bold	Entero largo	1
Italic	Entero largo	2
Underline	Entero largo	4

Ejemplo

Para probar si un elemento de menú se muestra en negrita, escribe:

```
If((Get menu item style($vlMenu;$vlItem)&Bold)#0)
   `...
End if
```

GET MENU ITEMS

GET MENU ITEMS (menu ; arrayTitMenus ; arraysRefMenus)

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Referencia de menú o número de menú
arrayTitMenus	Array cadena	⬅	Array de títulos de menú
arraysRefMenus	Array cadena	⬅	Array de referencias de menú

Descripción

*El comando **GET MENU ITEMS** devuelve, en los arrays arrayTitMenus y arraysRefMenus, los títulos e identificadores de todas las líneas de menú o de la barra de menús designada por el parámetro menu.*

*En el parámetro menu, puede pasar una referencia de menú (MenuRef), un número de barra de menús o una referencia de barra de menú obtenida utilizando el comando **Get menu bar reference**.*

Si ninguna referencia de menú está asociada a un elemento, una cadena vacía se devuelve en el elemento de array correspondiente.

Ejemplo

Usted quiere conocer el contenido de la barra de menú del proceso actual:

```
ARRAY STRING(32;arrayTitMenus;0)
ARRAY STRING(16;arraysRefMenus;0)
RefBarMenu:=Get menu bar reference(Frontmost process)
GET MENU ITEMS(RefBarMenu;arrayTitMenus;arraysRefMenus)
```

Get menu title

Get menu title (menu {; proceso}) -> Resultado

Parámetro	*Tipo*		*Descripción*
menu	*Entero largo, MenuRef*	→	*Número de menú o Referencia de menú*
proceso	*Entero largo*	→	*Número de referencia del proceso*
Resultado	*Cadena*	↩	*Título del menú*

Descripción

El comando ***Get menu title*** *devuelve el título del menú cuyo número o referencia se pasa en menu.*

Si omite el parámetro proceso, ***Get menu title*** *se aplica a la barra de menús del proceso actual. De lo contrario,* ***Get menu title*** *se aplica a la barra de menú para el proceso cuyo número de referencia se pasa en proceso.*

Nota: *si pasa un MenuRef en menu, el parámetro proceso es inútil y será ignorado.*

Get selected menu item parameter

Get selected menu item parameter -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Cadena*		*Parámetro personalizado de la línea de menú*

Descripción

*El comando **Get selected menu item parameter** devuelve la cadena de caracteres personalizada asociada a la línea de menú seleccionada. Este parámetro deber haber sido definido de antemano utilizando el comando **SET MENU ITEM PARAMETER**.*
Si ninguna línea de menú ha sido seleccionada, el comando devuelve una cadena vacía"".

INSERT MENU ITEM

INSERT MENU ITEM (menu ; depuesDe ; textoElem {; subMenu {; proceso}}{; *})

Parámetro	Tipo		Descripción
menu	Entero largo	➡	Número de menú o referencia de menú
depuesDe	Entero largo	➡	Número de la línea de menú
textoElem	Cadena	➡	Texto para la línea de menú a insertar
subMenu	MenuRef	➡	Referencia del submenú asociado con la línea
proceso	Entero largo	➡	Número de referencia del proceso
*	Operador	➡	Si se pasa: considerar metacaracteres como caracteres estándar

Descripción

*El comando **INSERT MENU ITEM** inserta nuevas líneas en el menú cuyo número o referencia se pasa en menu y las ubica después de la línea de menú cuyo número se pasa en despuesItem.*

*Si omite el parámetro proceso, **INSERT MENU ITEM** se aplica a la barra de menús del proceso actual. De lo contrario, **INSERT MENU ITEM** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

***Nota:** si pasa un MenuRef en menu, el parámetro proceso es inútil y se ignorará.*

*Si no pasa el parámetro *, INSERT MENU ITEM le permite insertar uno o varios comandos de menú en una sola llamada.*

***INSERT MENU ITEM** funciona como **APPEND MENU ITEM**, excepto que le permite insertar los comandos en cualquier parte del menú, mientras que **APPEND MENU ITEM** siempre los añade al final del menú.*

*Consulte la descripción del comando **APPEND MENU ITEM** para más información sobre la definición de los comandos de menús pasados en itemtext.*

***Nota:** la constante ak standard action title es soportada en el parámetro itemText (4D v16 R3 y superior).*

*El parámetro opcional submenu permite designar un menú como línea insertada y definir un submenú jerárquico. De pasar en este parámetro una referencia de menú (cadena de tipo MenuRef) especificando un menú creado, por ejemplo, utilizando el comando de menú **Create menu**. Si el comando añade más de una línea de menú, el submenú se asocia con la primera línea.*

***Importante:** las nuevas líneas no tienen métodos o acciones asociados. Deben asociarse utilizando los comandos **SET MENU ITEM PROPERTY** o **SET MENU ITEM METHOD**, o los elementos también pueden ser administrados desde un método de formulario utilizando el comando **Menu selected**.*

Ejemplo

El siguiente ejemplo crea un menú que consiste en dos comandos los cuales asignan un método:

```
MenuRef:=Crear menu
APPEND MENU ITEM(MenuRef;"Caracteres")
SET MENU ITEM METHOD(MenuRef;1;"CarMgmtDial")
INSERT MENU ITEM(MenuRef;1;"Parrafos")
SET MENU ITEM METHOD(MenuRef;2;"ParaMgmtDial")
```

Menu selected

Menu selected {(subMenu)} -> Resultado

Parámetro	Tipo		Descripción
subMenu	MenuRef	⇐	Referencia del menú que contiene la línea seleccionada
Resultado	Entero largo	⇒	Comando de menú seleccionado Palabra superior: Número de menú Palabra inferior: Número de comando de menú

Descripción

Menu selected *se utiliza sólo cuando se muestran formularios. Esta función detecta el comando de menú elegido en un menú y en el caso de un submenú jerárquico, devuelve la referencia del submenú.*

Consejo: *siempre que sea posible, utilice métodos asociados con comandos de menú en una barra asociada (con un número de barra negativo) en lugar de utilizar* ***Menu selected****. Las barras de menús asociadas son más fáciles de administrar, ya que no es necesario probar su selección. Sin embargo, si utiliza los comandos* ***APPEND MENU ITEM*** *o* ***INSERT MENU ITEM****, debe utilizar* ***Menu selected*** *porque las líneas de menús añadidas por estos comandos no tienen métodos asociados.*

El comando ***Menu selected*** *puede utilizarse para trabajar con submenús jerárquicos. Cuando se selecciona una línea de un menú jerárquico más allá del primer nivel, el comando devuelve, en el parámetro opcional submenú, la referencia (tipo MenuRef, cadena de 16 caracteres) del submenú al cual pertenece la línea seleccionada. Si el comando de menú no contiene un submenú jerárquico, este parámetro recibe una cadena vacía.*

Menu selected *devuelve el número del menú seleccionado, un entero largo. Para encontrar el número de menú, divida* ***Menu selected*** *por 65,536 y convierta el resultado en un entero. Para obtener el número del comando de menú, calcule el módulo de* ***Menu selected*** *con el coeficiente 65,536. Utilice las siguientes fórmulas para calcular el número de menú y del comando de menú:*

```
Menu:=Menu selected\65536
comando de menu:=Menu selected%65536
```

Igualmente puede extraer estos valores utilizando los ***Operadores bitwise*** *como en el siguiente ejemplo:*

```
Menu:=(Menu selected & 0xFFFF0000)>>16
comando de menu:=Menu selected & 0xFFFF
```

Si ningún comando de menú está seleccionado, ***Menu selected*** *devuelve 0.*

Ejemplo

El siguiente método de formulario utiliza ***Menu selected*** *para proporcionar los argumentos menú y línea de menú a* ***SET MENU ITEM MARK****:*

```
Case of
   :(Form event=On Menu Selected)
      C_STRING(16;$MenuRefIncludingItem)
      C_LONGINT($ref;$MenuNum;$MenuItemNum)
      $ref:=Menu selected($MenuRefIncludingItem)
      $MenuNum:=$ref\65536
      $MenuItemNum:=$ref%65536
      SET MENU ITEM MARK(MenuRefIncludingItem;$MenuItemNum;Char(18))
End case
```

Nota: *el evento de formulario* ***On Menu Selected*** *no es activado si ninguna línea está seleccionada, $MenuRefIncludingItem siempre es dado y $MenuNum vale 0 si el menú no es uno de los menús de la barra.*

RELEASE MENU

RELEASE MENU (menu)

Parámetro	Tipo		Descripción
menu	MenuRef	➡	Referencia de menú

Descripción

*El comando **RELEASE MENU** borra de la memoria el menú cuya referencia se pasa en menu.. La regla es la siguiente: a cada **Create menu** debe corresponder un **RELEASE MENU**.*

El comando elimina todas las instancias de menu en todas las barras de menú y todos los procesos. Si el menú pertenece a una barra de menú en uso, continuará funcionando pero no podrá ser modificada. Sólo se borrará realmente de la memoria cuando la última barra de menús en la que aparezca no esté más en uso.

Este comando puede utilizarse con menús usados como barras de menús.

*Los submenús utilizados por menu no se borran si fueron creados utilizando el comando **Create menu**. En este caso, debe eliminarlos individualmente (ver la regla mencionada anteriormente). Sin embargo, si los submenús vienen de la duplicación de un menú existente, no llame **RELEASE MENU** con sus instancias porque 4D las borrará automáticamente.*

Ejemplo

Este ejemplo muestra las diferentes formas de utilizar este comando:

```
newMenu:=Create menu
APPEND MENU ITEM(newMenu;"Test1")
subMenu:=Create menu
APPEND MENU ITEM(subMenu;"SubTest1")
APPEND MENU ITEM(subMenu;"SubTest2") // Creación de elementos en el submenú

APPEND MENU ITEM(newMenu;"Test2";subMenu) // Asociar los submenús al menú

 //En este momento, el submenú se asocia al menú y si no lo necesitamos más adelante, este es el momento adecuado para borrarlo.
 //Al borrrarlo no se borra inmediatamente subMenu ya que aún está siendo utilizado por newMenu. subMenu sólo se borra cuando se borra newMenu.
 //Borrar el submenú en este momento le permite ahorrar memoria
RELEASE MENU(subMenu)

$result1:=Dynamic pop up menu(newMenu) //Uso de menú
copyMenu:=Create menu(newMenu) // Creación de un menú por copia de newMenu (y duplicación de subMenu)
RELEASE MENU(newMenu) // Ya no necesitamos a newMenu.

$result2:=Dynamic pop up menu(copyMenu)
RELEASE MENU(copyMenu)
 //No debe preocuparse por borrar los submenús de copyMenu ya que no se creó directamente utilizando Create menu
 //La regla a seguir es: cada Create menu debe tener un RELEASE MENU correspondiente
```

SET MENU BAR

SET MENU BAR (barra {; proceso}{; *})

Parámetro	Tipo		Descripción
barra	Entero largo, Cadena, MenuRef	➡	Número o nombre de la barra de menús o Referencia de menú
proceso	Entero largo	➡	Número de referencia del proceso
*	Operador	➡	Guardar el estado de la barra de menús

Descripción

MENU BAR reemplaza la barra de menús actual con la especificada por barra en el proceso actual únicamente. En el parámetro barra, puede pasar el número o nombre de la nueva barra. Igualmente puede pasar una referencia única de menú (tipo MenuRef, cadena de 16 caracteres). Cuando trabaja con referencias, los menús pueden ser utilizados como barras de menú y viceversa (ver la sección **Gestión de menús**).

Nota: el nombre de una barra de menús puede contener hasta 31 caracteres y debe ser único.

Si pasa el parámetro opcional proceso cambia la barra de menús del proceso especificado por barra.

Nota: si pasa un parámetro MenuRef en barra, el parámetro proceso es inútil y se ignorará.

El parámetro opcional * le permite conservar el estado de la barra de menús. Si este parámetro se omite, **MENU BAR** reinicializa la barra de menús cuando el comando se ejecuta.

Por ejemplo, imagine que **SET MENU BAR(1)** se ejecuta. Luego, varios comandos de menú se desactivan utilizando el comando **DISABLE MENU ITEM**.

Si **SET MENU BAR(1)** se ejecuta una segunda vez, desde el mismo proceso o desde un proceso diferente, todos los comandos de menú regresarán a su estado de activación inicial.

Si **SET MENU BAR(1;*)** se ejecuta, la barra de menús conservará su estado anterior, y los comandos de menú que estaban inactivos permanecerán inactivos.

Nota: si pasa un parámetro MenuRef en barra, el parámetro * es inútil y se ignorará.

Cuando un usuario entra al entorno Aplicación, se muestra la primera barra de menús (Barra #1). Puede cambiar esta barra de menús por defecto abriendo la base y especificando la barra de menús deseada en el **Método de base On Startup** o en el método de inicio asociado a un usuario individual.

Ejemplo 1

El siguiente ejemplo cambia la barra de menús actual por la barra de menús #3 y restablece el estado de los comandos de menú a sus estados originales:

```
SET MENU BAR(3)
```

Ejemplo 2

El siguiente ejemplo cambia la barra de menús actual por la barra de menús llamada "BarraForm1" y guarda el estado de los comandos de menús. Comandos de menús que fueron desactivados previamente aparecerán inactivos.

```
SET MENU BAR("BarraForm1";*)
```

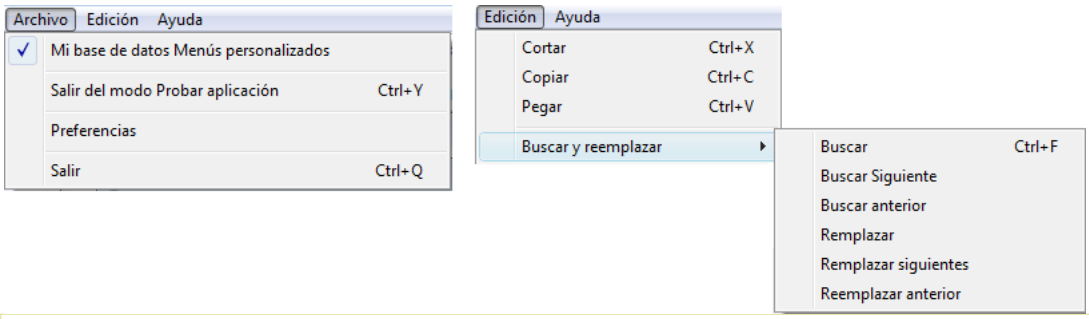
Ejemplo 3

El siguiente ejemplo reemplaza la barra de menús actual por la barra de menús #3 mientras los registros son modificados. Una vez los registros sean modificados, la barra de menús #2 se restaura. El estado de los comandos de menú se conserva:

```
SET MENU BAR(3)
ALL RECORDS([Clientes])
MODIFY SELECTION([Clientes])
SET MENU BAR(2;*)
```

Ejemplo 4

En este ejemplo, crearemos por programación, una barra de menús incluyendo los siguientes menús Archivo y Edición:




```
  `Metodo de creación menú Archivo
C_TEXT(16;MenuArchivo) ` MenuArchivo contendrá la referencia de menu Archivo
MenuArchivo:=Create menu
INSERT MENU ITEM(MenuArchivo;-1;"Mi base de datos "+Get indexed string(131;29))
SET MENU ITEM MARK(MenuArchivo;1;Char(18))
INSERT MENU ITEM(MenuArchivo;-1;"(-")
INSERT MENU ITEM(MenuArchivo;-1;"Salir del modo Test Application mode/Y")
SET MENU ITEM PROPERTY(MenuArchivo;3;Associated standard action;ak return to design mode)
INSERT MENU ITEM(MenuArchivo;-1;"(-")
INSERT MENU ITEM(MenuArchivo;-1;"Preferencias")
SET MENU ITEM PROPERTY(MenuArchivo;5;Associated standard action;ak database settings) `Preferencias
INSERT MENU ITEM(MenuArchivo;-1;"(-")
INSERT MENU ITEM(MenuArchivo;-1;Get indexed string(131;30))
SET MENU ITEM PROPERTY(MenuArchivo;7;Associated standard action;ak quit) `Salir
SET MENU ITEM SHORTCUT(MenuArchivo;7;Character code("Q"))

  `Método de creación del menú Buscar y reemplazar
C_TEXT(MenuBuscaryReemplazar) `MenuBuscaryReemplazar contendrá la referencia del menú Buscar y reemplazar
MenuBuscaryReemplazar:=Create menu
APPEND MENU ITEM(MenuBuscaryReemplazar;"Buscar;Buscar Siguiente;Buscar anterior;(-;Remplazar;Remplazar siguientes;Reemplazar
anterior")
SET MENU ITEM SHORTCUT(MenuBuscaryReemplazar;1;Character code("F"))
SET MENU ITEM SHORTCUT(MenuBuscaryReemplazar;5;Character code("R"))
SET MENU ITEM METHOD(MenuBuscaryReemplazar;1;"MiMetodoBuscar")

  `Método de creación del menú Edición
C_TEXT(MenuEdicion) `EditMenu contendrá la referencia del menú Edición
MenuEdicion:=Create menu
APPEND MENU ITEM(MenuEdicion;"Cortar;Copiar;Pegar")
SET MENU ITEM SHORTCUT(MenuEdicion;1;Character code("X"))
SET MENU ITEM PROPERTY(MenuEdicion;1;Associated standard action;ak cut)
SET MENU ITEM SHORTCUT(MenuEdicion;2;Character code("C"))
SET MENU ITEM PROPERTY(MenuEdicion;2;Associated standard action;ak copy)
SET MENU ITEM SHORTCUT(MenuEdicion;3;Character code("V"))
SET MENU ITEM PROPERTY(MenuEdicion;3;Associated standard action;ak paste)
INSERT MENU ITEM(MenuEdicion;-1;"(-")
INSERT MENU ITEM(MenuEdicion;-1;"Buscar y reemplazar";MenuBuscaryReemplazar) ` línea que tendrá el submenú

main_Bar:=Create menu ` Cree la barra de menús con los otros menús
INSERT MENU ITEM(main_Bar;-1;Get indexed string(79;1);MenuArchivo)
APPEND MENU ITEM(main_Bar;"Edición";MenuEdición)

SET MENU BAR(main_Bar)
```

SET MENU ITEM

SET MENU ITEM (menu ; lineamenu ; textoElem {; proceso}{; *})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Número de menú o referencia de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 para el último elemento añadido
textoElem	Cadena	➡	Nuevo texto de la línea de menú
proceso	Entero largo	➡	Número de referencia del proceso
*	Operador	➡	Si se pasa: considerar metacaracteres como caracteres estándar

Descripción

*El comando **SET MENU ITEM** cambia el texto de la línea de menú cuyo número o referencia de menú se pasa en menu y cuyo número de elemento se pasa en menuItem, para el texto que pasó en itemText. Puede pasar -1 en menuItem para designar el último elemento añadido a menu.*

*Si no pasa el parámetro *, los caracteres "especiales" incluídos en itemText (tales como (; o !) serán considerados como caracteres de instrucción (metacaracteres). Por ejemplo, para definir un elemento de menú como una línea de separación, inserte el carácter "-" en itemText. Si pasa el parámetro **, los caracteres "especiales" se considerarán como caracteres estándar. Por favor consulte la descripción del comando **APPEND MENU ITEM** para más información sobre los detalles de estos caracteres.*

*Si omite el parámetro proceso, **SET MENU ITEM** se aplica a la barra de menús del proceso actual. De lo contrario, **SET MENU ITEM** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

***Nota:** si pasa un MenuRef en menu, el parámetro proceso es inútil y se ignorará.*

SET MENU ITEM ICON

SET MENU ITEM ICON (menu ; lineamenu ; refIcon {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Referencia de menú o número de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 para el último elemento añadido
refIcon	Texto, Entero largo	➡	Nombre o número de la librería de imágenes a asociar a la línea de menú
proceso	Entero largo	➡	Número de proceso

Descripción

*El comando **SET MENU ITEM ICON** permite modificar el icono asociado a la línea de menú designada por los parámetros menu y lineaMenu.*
Puede pasar -1 en lineaMenu para especificar el último elemento añadido a menu.

En menu, puede pasar una referencia de menú (MenuRef) o un número de menú. Si pasa una referencia de menú, el comando aplicará a todas las instancias del menú en todos los procesos. En este caso, si se pasa el parámetro proceso se ignora. Si pasa un número de menú, el comando tendrá en cuenta el menú correspondiente en la barra de menús principal del proceso actual. Si quiere designar otro proceso, pase su número en el parámetro opcional proceso.

En refIcon, puede pasar el nombre o el número de la imagen de la librería a utilizar como icono. Puede utilizar una imagen de la librería o una referencia de imagen.
• Imagen de librería: puede pasar el nombre o el número de imagen. Por lo general es preferible utilizar su número en lugar del nombre ya que los números de imagen son identificadores únicos, que no es el caso con de los nombres.
*• Referencia de imagen: la imagen debe estar ubicada en la carpeta **Resources** de la base y debe utilizar una sintaxis "archivo:{nombreruta}NombreArchivo" en refIcon. Para mayor información sobre la carpeta **Resources**, consulte la sección **Recursos**.*

Ejemplo

Uso de una imagen ubicada en la carpeta Resources de la base:

```
SET MENU ITEM ICON($MenuRef;2;"File:ES.lproj/spot.png")
```

SET MENU ITEM MARK

SET MENU ITEM MARK (menu ; lineamenu ; marca {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Número de menú o Referencia de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 por el último elemento añadido
marca	Cadena	➡	Nueva marca de línea de menú
proceso	Entero largo	➡	Número de referencia de proceso

Descripción

*El comando **SET MENU ITEM MARK** cambia la marca del elemento de menú cuyo número o referencia de menú se pasa en menu y cuyo número de línea se pasa en menuItem al primer carácter de la cadena pasada en marca. Puede pasar -1 en menuItem para designar la última línea añadida al menú.*

*Si omite el parámetro proceso, **SET MENU ITEM MARK** se aplica a la barra de menús del proceso actual. De lo contrario, **SET MENU ITEM MARK** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

***Nota**: si pasa un parámetro MenuRef en menú, el parámetro proceso es inútil y se ignorará*

Si pasa una cadena vacía, toda marca de la línea de menú se elimina. De lo contrario:

- *En Macintosh, el primer carácter de la cadena se convierte en la marca de la línea de menú. Generalmente, pasará **Char (18)**, el cual es el carácter de marca para los menús Macintosh.*
- *En Windows, la marca estándar de Window se asocia al menú.*

Ejemplo

*Ver ejemplo para el comando **Get menu item mark**.*

SET MENU ITEM METHOD

SET MENU ITEM METHOD (menu ; lineaMenu ; nomMetodo {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Referencia de menú o número de menú
lineaMenu	Entero largo	➡	Número de línea de menú o -1 para el último elemento añadido al menú
nomMetodo	Cadena	➡	Nombre del método
proceso	Entero largo	➡	Número de proceso

Descripción

El comando **SET MENU ITEM METHOD** puede utilizarse para modificar el método de proyecto 4D asociado a la línea de menú designada por los parámetros menu y lineaMenu.

Puede pasar -1 en lineaMenu para especificar la última línea a añadida a menu.

En menu, puede pasar una referencia de menú (MenuRef) o un número de menú. Si pasa una referencia de menú, el comando se aplicará a todas las instancias del menú en todos los procesos En este caso, si se pasa el parámetro proceso se ignora. Si pasa un número de menú, el comando se aplicará al menú correspondiente en la barra de menús principal del proceso actual. Si quiere designar otro proceso, pase su número en el parámetro opcional proceso.

En metodo, pase el nombre del método 4D como cadena de caracteres (expresión).

Nota: si la línea de menú corresponde al título de un submenú jerárquico, el método no se llamará cuando la línea de menú sea seleccionada.

Ejemplo

Consulte el ejemplo del comando **SET MENU BAR**.

SET MENU ITEM PARAMETER

SET MENU ITEM PARAMETER (menu ; lineaMenu ; param)

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Referencia de menú o número de menú
lineaMenu	Entero largo	➡	Número de línea de menú o -1 para la última línea añadida al menú
param	Cadena	➡	Cadena a asociar como parámetro

Descripción

*El comando **SET MENU ITEM PARAMETER** permite asociar una cadena de caracteres personalizada con una línea de menú designada por los parámetros menu y lineaMenu.*

*Este parámetro es utilizado principalmente por el comando **Dynamic pop up menu**.*

Ejemplo

Este código ofrece un menú que incluye los nombres de las ventanas abiertas y permite recuperar el número de la ventana elegida:

```
WINDOW LIST($alWindow)
$tMenuRef:=Create menu
For($i;1;Size of array($alWindow))
   APPEND MENU ITEM($tMenuRef;Get window title($alWindow{$i})) // Título de la línea del menú
   SET MENU ITEM PARAMETER($tMenuRef;-1;String($alWindow{$i})) // Valor devuelto por la línea del menú
End for
$tWindowRef:=Dynamic pop up menu($tMenuRef)
RELEASE MENU($tMenuRef)
```

SET MENU ITEM PROPERTY

SET MENU ITEM PROPERTY (menu ; lineaMenu ; propiedad ; valor {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Referencia del menú o número de menú
lineaMenu	Entero largo	➡	Número de línea del menú o -1 para la última línea añadida al menú
propiedad	Cadena	➡	Tipo de propiedad
valor	Texto, Número, Booleano	➡	Valor de la propiedad
proceso	Entero largo	➡	Número del proceso

Descripción

*El comando **SET MENU ITEM PROPERTY** permite fijar el valor de la propiedad para la línea de menú designada por los parámetros menu y lineaMenu.*

Puede pasar -1 en lineaMenu para especificar la última línea añadida a menu.

En menu, puede pasar una referencia de menú (MenuRef) o un número de menú. Si pasa una referencia de menú, el comando se aplicará a todas las instancias del menú en todos los procesos En este caso, si se pasa el parámetro proceso se ignora. Si pasa un número de menú, el comando se aplicará al menú correspondiente en la barra de menús principal del proceso actual. Si quiere designar otro proceso, pase su número en el parámetro opcional proceso.

*En el parámetro propiedad, pase la propiedad cuyo valor quiere modificar y pase el nuevo valore en valor. Para el parámetro propiedad, puede utilizar una **propiedad estándar** (una de las constantes del tema "**Propiedades de ítem de menú**") o una **propiedad personalizada**:*

- ***Propiedad estándar**: las constantes del tema "**Propiedades de ítem de menú**" así como sus posibles valores son descritas a continuación.*

Constante	Tipo	Valor	Comentario
Access privileges	Cadena	4D_access_group	Activar la opción "Iniciar un nuevo proceso" 0 = No, 1 = Sí
Associated standard action	Cadena	4D_standard_action	Asociar una acción estándar a la línea de menú Ver las constantes del tema "**Acción estándar**"
Start a new process	Cadena	4D_start_new_process	Asignar un grupo de acceso al comando 0 = Sin restricción >0 = Número de grupo

  *En el caso de la propiedad Associated standard action, puede pasar en el parámetro valor un nombre de acción estándar. Para obtener una lista completa de las acciones disponibles, consulte la sección **Acciones estándar** del manual de Diseño. Las acciones más comunes también están disponibles como constantes en el tema **Acción estándar**.*
  ***Nota de compatibilidad:** en versiones anteriores, las constantes del tema **Valores para acción estándar asociada** se utilizaron en el parámetro valor (Entero largo). A partir de 4D v16 R3, son obsoletas, pero aún son soportadas por compatibilidad.*

  ***Nota:** si la línea de menú corresponde al título de un submenú jerárquico, la acción estándar no se llamará cuando se seleccione la línea de menú.*

- ***Propiedad personalizada**: en propiedad, puede pasar todo texto personalizado y asociar un valor de tipo texto, numérico o booleano. Este valor será almacenado con el elemento y puede ser recuperado utilizando el comando **GET MENU ITEM PROPERTY**. En el parámetro propiedad puede utilizar toda cadena personalizada, simplemente asegúrese de utilizar un título utilizado por 4D (por convención, las propiedades definidas por 4D comienzan por "4D_").*

SET MENU ITEM SHORTCUT

SET MENU ITEM SHORTCUT (menu ; lineamenu ; tecla ; modificadores {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Referencia de menú o número de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 para la última línea añadida al menú
tecla	Cadena, Entero largo	➡	Código del carácter de atajo de teclado o letra del atajo de teclado
modificadores	Entero largo	➡	Modificador(es) a asociar al atajo (se ignora si se pasa el código de tecla)
proceso	Entero largo	➡	Número de referencia del proceso

Descripción

*El comando **SET MENU ITEM SHORTCUT** cambia el atajo Ctrl (Windows) o comando (Macintosh) para el comando de menú cuyos números de menú y de elemento se pasan en menu y menuItem, por el carácter cuyo carácter de código o texto se pasa en itemKey. Puede pasar -1 en menuItem para indicar el último elemento añadido al menú. Esta tecla se combinará automáticamente con la tecla **Ctrl** (Windows) o **Comando** (Macintosh) para definir el nuevo atajo de teclado.*

*Puede pasar directamente el nombre de la tecla como texto (una letra) en el parámetro itemKey, por ejemplo "U" para especificar el atajo **Ctrl+U** (Windows) o **Comando+U** (Mac OS). Cuando utilice esta sintaxis, también puede pasar el parámetro opcional modificadores para asociar los modificadores adicionales al atajo. De esta forma puede definir los atajos de tipo **Ctrl+Alt+Mayús+Z** (Windows) o **Cmd+Opción+Mayús+Z** (Mac OS).*

Para hacer esto, pase en modificadores los siguientes valores:

- *256 para la tecla **Comando** (Mac OS) o **Ctrl** (Windows)*
- *512 para la tecla **Mayús***
- *2048 para la tecla **Opcion** (Mac OS) o **Alt** (Windows)*
- *Para asociar ambas teclas, combine sus valores.*

***Nota:** puede definir el valor a pasar utilizando las constantes Shift key mask y Option key mask del tema " ".*

*La tecla **Ctrl** (Windows) y **Comando** (Mac OS) son añadidas automáticamente por 4D al atajo de teclado, sin importar si lo indicó explícitamente en el parámetro modifiers. De manera que no necesita añadir el valor 256 a este parámetro, a menos que la tecla sea el único modificador, en ese caso debe pasar el valor 256 o la constante correspondiente en modifiers.*

***Nota:** por compatibilidad, el comando también acepta un código de carácter como parámetro tecla (sintaxis anterior). En este caso, el parámetro modificadores no se tiene en cuenta y puede omitirse. El atajo sólo se asocia a la tecla **Ctrl** (Windows) o **Comando** (Mac OS).*

El parámetro modificadores no se tiene en cuenta si la tecla de modificación está definida vía su condigo de carácter (sintaxis anterior).

*Si omite el parámetro proceso, **SET MENU ITEM SHORTCUT** se aplica a la barra de menús del proceso actual. De lo contrario, **SET MENU ITEM SHORTCUT** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

***Nota:** si pasa un parámetro MenuRef en menú, el parámetro proceso es inútil y se ignorará.*

Si pasa 0 (cero) en tecla, todo atajo se elimina del elemento de menú.

Ejemplo 1

Definición del atajo Ctrl+Mayús+U (Windows) y Cmd+Mayús+U (Mac OS) para la línea "Subrayado":

```
SET MENU ITEM(MenuRef;1;"Subrayado")
SET MENU ITEM SHORTCUT(MenuRef;1;"U";Shift key mask)
```

Ejemplo 2

Definición del atajo Ctrl+R (Windows) y Cmd+R (Mac OS) para el elemento de menú "Reiniciar":

```
INSERT MENU ITEM(FileMenu;-1;"Reiniciar")
SET MENU ITEM SHORTCUT(FileMenu;-1;"R";Command key mask)
```

SET MENU ITEM STYLE

SET MENU ITEM STYLE (menu ; lineamenu ; estiloItem {; proceso})

Parámetro	Tipo		Descripción
menu	Entero largo, MenuRef	➡	Número de menú o Referencia de menú
lineamenu	Entero largo	➡	Número de línea de menú o -1 para el último elemento añadido
estiloItem	Entero largo	➡	Nuevo estilo de la línea de menú
proceso	Entero largo	➡	Número de referencia de proceso

Descripción

*El comando **SET MENU ITEM STYLE** cambia el estilo de la fuente de la línea de menú cuyo número o referencia de menú se pasa en menu y cuyo número de elemento se pasa en menuItem de acuerdo al estilo de fuente pasado en itemEstilo. Puede pasa -1 en menuItem para indicar el último elemento añadido a menu.*

*Si omite el parámetro proceso, **SET MENU ITEM STYLE** se aplica a la barra de menús del proceso actual. De lo contrario, **SET MENU ITEM STYLE** se aplica a la barra de menús del proceso actual cuyo número de referencia se pasa en proceso.*

***Nota:** si pasa un MenuRef en menu, el parámetro proceso es inútil y se ignorará.*

En el parámetro itemEstilo puede definir el estilo del elemento. Pase una combinación (una o una suma) de las siguientes constantes predefinidas:

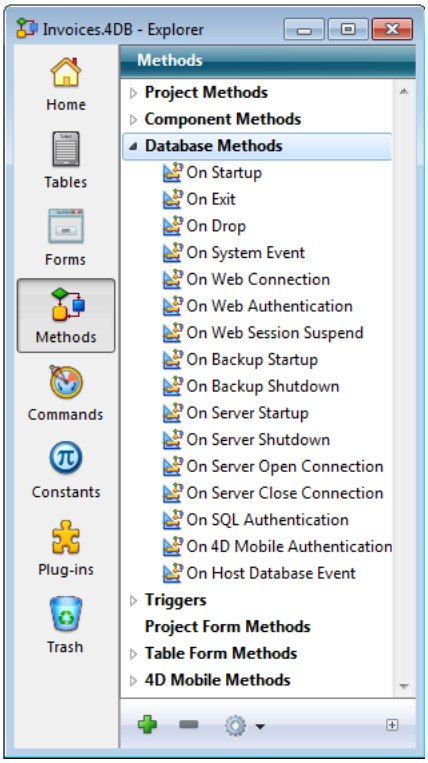
Constante	Tipo	Valor
Plain	Entero largo	0
Bold	Entero largo	1
Italic	Entero largo	2
Underline	Entero largo	4

Métodos base

- *Métodos de base de datos*
- *Método base On 4D Mobile Authentication*
- *Método base On Backup Shutdown*
- *Método base On Backup Startup*
- *Método base On Drop*
- *Método base On Exit*
- *Método base On Host Database Event*
- *Método base On Server Close Connection*
- *Método base On Server Open Connection*
- *Método base On Server Shutdown*
- *Método base On Server Startup*
- *Método base On SQL Authentication*
- *Método base On Startup*
- *Método base On System Event*
- *Método base On Web Authentication*
- *Método base On Web Close Process*
- *Método base On Web Connection*

Métodos de base de datos

Los métodos de base de datos son métodos ejecutados automáticamente por 4D cuando ocurre un evento de sesión general.



Para crear, abrir o editar un método de base de datos:

*1. Abra la ventana del **Explorador**.*
*2. Seleccione la página **Métodos**.*
*3. Despliegue el tema **Métodos de base de datos**.*
4. Haga doble clic en el método.

o:

1. Seleccione el método.
2. Presione Intro o Retorno de carro.

Un método base se edita de la misma forma que cualquier otro método.

No es posible llamar un método de base desde otro método. Los métodos de base se invocan automáticamente por 4D en ciertos puntos de una sesión de trabajo. La siguiente tabla resume la ejecución de los métodos de base:

Método de base	***4D local***	***4D Server***	***4D remoto***
On Startup	*Sí, una vez*	*No*	*Sí, una vez*
On Exit	*Sí, una vez*	*No*	*Sí, una vez*
On Drop	*Sí, varias veces*	*No*	*Sí, varias veces*
On System Event	*Sí, varias veces*	*Sí, varias vecesYes, Multiple*	*Sí, varias veces*
On Web Connection	*Sí, varias veces*	*Sí, varias veces*	*Sí, varias veces*
On Web Authentication	*Sí, varias veces*	*Sí, varias veces*	*No*
On Web Session Suspend	*Sí, varias veces*	*Sí, varias veces*	*Sí, varias veces*
On Backup Startup	*Sí, varias veces*	*Sí, varias veces*	*Sí, varias veces*
On Backup Shutdown	*Sí, varias veces*	*Sí, varias veces*	*Sí, varias veces*
On Server Startup	*No*	*Sí, una vez*	*No*
On Server Shutdown	*No*	*Sí, una vez*	*No*
On Server Open Connection	*No*	*Sí, varias veces*	*No*
On Server Close Connection	*No*	*Sí, varias veces*	*No*
On SQL Authentication	*Sí, varias veces*	*Sí, varias veces*	*Sí, varias veces*
On 4D Mobile Authentication	*Sí, varias veces*	*Sí, varias veces*	*No*
On Host Database Event	*Sí, varias veces*	*Sí, varias veces*	*Sí, varias veces*

Método base On 4D Mobile Authentication

$1, $2, $3 -> Método base On 4D Mobile Authentication -> Resultado

Parámetro	Tipo		Descripción
$1	Texto	⬅	Nombre de usuario
$2	Texto	⬅	Contraseña
$3	Booleano	⬅	True = modo Digest, False = modo Basic
Resultado	Booleano	➲	True = petición aceptada, False = petición rechazada

Descripción

*El **Método base On 4D Mobile Authentication** le permite controlar de forma personalizada la apertura de las sesiones 4D Mobile (vía REST) en 4D. Este método base se destina principalmente al filtrado de conexiones cuando se establece una conexión entre un Wakanda Server y 4D.*

*Cuando la solicitud de apertura de sesión 4D Mobile proviene de Wakanda Server por medio del método **mergeOutsideCatalog()** (caso general), los identificadores de conexión están en el encabezado de la solicitud. El **Método base On 4D Mobile Authentication** se llama para que pueda evaluar estos identificadores. Puede utilizar la lista de usuarios de la base 4D o puede utilizar su propia tabla de identificadores.*

***Importante**: cuando **Método base On 4D Mobile Authentication** está definido (es decir, cuando contiene código), 4D le delega plenamente el control de las solicitudes 4D Mobile: cualquier ajuste realizado utilizando el menú "Lectura/Escritura" de la página Web/4D Mobile de las Propiedades de la base se ignora (ver el manual de Diseño).*
El método base recibe dos parámetros de tipo de texto ($1 y $2) y un valor booleano ($3), pasado por 4D, y devuelve un booleano, $0. Debe declarar estos parámetros de la siguiente manera:

```
//Método base On 4D Mobile Authentication
C_TEXT($1;$2)
C_BOOLEAN($0;$3)
... // Código para el método
```

$1 contiene el nombre del usuario y $2 la contraseña utilizada para la conexión.
La contraseña ($2) puede recibirse en claro o en forma hash, dependiendo del modo utilizado para la petición. Este modo es indicado por el parámetro $3 para permitirle realizar el procesamiento apropiado:

- *Si la contraseña se envía en claro (modo Basic), $3 devuelve **False**.*
- *If it is sent in hashed form (modo Digest), $3 devuelve **True**.*

Cuando la solicitud de conexión 4D Mobile proviene de Wakanda Server, la contraseña se envía siempre en forma de hash.

*Debe controlar los identificadores de la conexión 4D Mobile en el método base. Por lo general , se comprueba el nombre y la contraseña utilizando una tabla de usuarios personalizada. Si los identificadores son válidos, pase **True** en $0. La solicitud se acepta; 4D la ejecuta y devuelve el resultado en JSON.*
*De lo contrario, pase **False** en $0, en este caso, la conexión se rechaza y el servidor devuelve un error de autenticación al remitente de la petición.*

Si el usuario es referenciado en la lista de usuarios 4D de la base, puede comprobar la contraseña directamente a través de la siguiente instrucción:

```
$0:=Validate password($1;$2;$3)
```

*El comando **Validate password** se ha extendido para aceptar un nombre de usuario como primer parámetro, así como un parámetro opcional que indica si la contraseña se expresa en forma hash.*

*Si desea utilizar su propia lista de usuarios externos para la lista de la base 4D, puede guardar sus contraseñas en forma hash utilizando el mismo algoritmo que el utilizado por Wakanda Server cuando se envía la solicitud de conexión a **Método base On 4D Mobile Authentication** en $2. Para generar el hash para una contraseña utilizando este método, puede escribir:*

```
$HashedPasswd :=Generate digest($ClearPasswd ;4D digest)
```

*El comando **Generate digest** acepta 4D digest como algoritmo de hashing, correspondiente al método utilizado por 4D para su gestión interna de palabras claves.*

Ejemplo 1

Este ejemplo sólo acepta el usuario "admin" con la contraseña "123" que no corresponde a un usuario 4D:

```
//Método base On REST Authentication
C_TEXT($1;$2)
C_BOOLEAN($0;$3)
//$1: usuario
```

```
  //$2: contraseña
  //$3: modo digest
If($1="admin")
    If($3)
        $0:=($2=Generate digest("123";4D digest))
    Else
        $0:=($2="123")
    End if
Else
    $0:=False
End if
```

Ejemplo 2

*Este ejemplo de **Método base On 4D Mobile Authentication** verifica que la demanda de conexión proveniente de uno de los dos servidores Wakanda autorizados, guardados en los usuarios de la base 4D:*

```
C_TEXT($1;$2)
C_BOOLEAN($0)
ON ERR CALL("4DMOBILE_error")
If($1="WAK1")|($1="WAK2")
    $0:=Validate password($1;$2;$3)
Else
    $0:=False
End case
```

Método base On Backup Shutdown

$1 -> Método base On Backup Shutdown

Parámetro	Tipo		Descripción
$1	Entero largo	⇐	0 = backup ejecutado correctamente; otro valor = error, interrumpido por el usuario o código devuelto por On Backup Startup

---

*El **Método base On Backup Shutdown** se llama cada vez que termina un backup de la base. Las razones para detener un backup pueden ser el fin de la copia, la interrupción por parte del usuario o un error.*
Esto concierne a todos los entornos 4D (todos los modos), 4D Server así como las aplicaciones 4D compiladas y fusionadas con 4D Volume Desktop.

*El **Método base On Backup Shutdown** permite verificar que el backup fue ejecutado correctamente. El método recibe, en el parámetro $1, un valor indicando el estado del backup una vez terminado:*

- *Si el backup se ejecutó correctamente, $1 es igual a 0.*
- *Si el backup fue interrumpido por el usuario o por un error, $1 es diferente de 0.*
    - *Si el backup fue detenido por el **Método de base de datos On Backup Startup** ($0 # 0), $1 obtiene el valor devuelto en el parámetro $0. Esto le permite implementar un sistema de gestión de errores personalizado.*
    - *Si el backup fue detenido por un error, el código del error se devuelve en $1.*

*En todos los casos, puede obtener información sobre el error utilizando el comando **GET BACKUP INFORMATION**.*

***Nota**: debe declarar el parámetro $1 (entero largo) en el método de la base:*

```
C_LONGINT($1)
```

*Es importante notar que en caso de error durante el backup (disco lleno, soporte inaccesible, etc.), la información relativa al error se muestra únicamente en el monitor de 4D Server o en el CSM, y se copia en el historial de backups. No se muestra una caja de diálogo de alerta y la variable error no se modifica. Si quiere notificar al administrador que se produjo un error, particularmente en el contexto de una aplicación en modo cliente/servidor, es necesario utilizar el **Método base On Backup Shutdown**.*

Método base On Backup Startup

Método base On Backup Startup -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entero largo*		*0 = backup autorizado; valor diferente de 0 = backup no autorizado*

---

*El **Método base On Backup Startup** se llama cada vez que un backup está a punto de iniciar (backup manual, backup automático programado, o utilizando el comando **BACKUP**).*

Esto concierne a todos los entornos 4D: 4D en modo local, 4D Server, 4D en modo remoto, 4D Desktop y bases fusionadas con 4D Desktop.

*El **GET BACKUP INFORMATION** permite verificar el inicio del backup. En este método, debe devolver en el parámetro $0 un valor que autorice o rechace el backup:*

- *Si $0 = 0, el backup puede comenzar.*
- *Si $0 # 0, el backup no es autorizado. La operación se cancela y devuelve un error. Puede obtener el error utilizando el comando **Método base On Backup Startup**.*

Puede utilizar este método base para verificar las condiciones de ejecución del backup (usuario, fecha del último, etc.).

***Nota:** debe declarar el parámetro $0 (entero largo) en el método de la base:*

```
C_LONGINT($0).
```

Método base On Drop

Método base On Drop
Este comando no requiere parámetros

---

*El **Método base On Drop** está disponible en aplicaciones 4D locales o remotas.*

Este método de base se ejecuta automáticamente en el caso de soltar objetos en la aplicación 4D fuera de un contexto formulario o dialogo. Dependiendo de la plataforma y del tipo de aplicación, se soportan diferentes tipos de soltar:

Acción	***Plataforma(s)***	***Comentario***
Soltar en un área vacía de la ventana MDI	*Windows*	*No disponible cuando la base se ejecuta en modo SDI ya que no hay ninguna ventana MDI en este contexto (consulte la sección **Modo SDI en Windows**).*
Soltar en el icono 4D en el Dock	*macOS*	
Soltar en el icono 4D en el escritorio del sistema	*Windows(*) y macOS*	***Método base On Drop** se llama únicamente, si la aplicación ya ha sido lanzada, excepto en el caso de las aplicaciones fusionadas con 4D Desktop. En este caso, el método base se llama incluso cuando la base no ha sido lanzada. Esto significa que es posible definir firmas de documentos personalizadas.*

(*) No compatible con 4D Developer 64 bits en Windows porque esta acción inicia una nueva instancia de aplicación (funcionalidad del sistema).

*Bajo Mac OS, debe presionar las teclas **Opción**+**Comando** al soltar para que se llame el método de base.*

Ejemplo

Este ejemplo puede ser utilizado para abrir un documento 4D Write que fue soltado fuera de un formulario:

```
`Método base On Drop
archivoSoltado:=Get file from pasteboard(1)
If(Position(".4W7";droppedFile)=Length(archivoSoltado)-3)
    areaExterna:=Open external window(100;100;500;500;0;archivoSoltado;"_4D Write")
    WR OPEN DOCUMENT(areaExterna;archivoSoltado)
End if
```

Método base On Exit

Método base On Exit
Este comando no requiere parámetros

---

*El **Método base On Exit** se llama una vez cuando sale de la base.*
Este método se utiliza en los siguientes entornos 4D:

- *4D en modo local*
- *4D en modo remoto (del lado del cliente)*
- *Aplicación 4D compilada y fusionada con 4D Volume Desktop*

***Nota**: el **Método base On Exit** NO es invocado por 4D Server.*
*El **Método base On Exit** es invocado automáticamente por 4D; a diferencia de los métodos de proyecto, usted no puede llamar este método por programación. Sin embargo, puede ejecutarlo desde el editor de métodos. Igualmente puede utilizar subrutinas.*
Se sale de una base si:

- *El usuario selecciona el comando **Salir** del menú **Archivo** en el entorno Diseño o desde el entorno Aplicación (Quit standard action).*
- *Se efectúa una llamada al comando **QUIT 4D.***
- *Un plug-in 4D hace una llamada al punto de entrada **QUIT 4D.***

Sin importar cómo se inicie la salida de la base, 4D realiza las siguientes acciones:

- *Si no existe un **Método base On Exit**, 4D aborta cada proceso en ejecución uno por uno, sin distinción. Si el usuario está introduciendo datos, los registros no se guardarán.*
- *Si existe un **Método base On Exit**, 4D comienza la ejecución de este método en un proceso local creado recientemente. Observe que saldrá eventualmente de 4D, el **Método base On Exit** puede realizar toda la limpieza o cierre de las operaciones que usted quiera, pero no puede negarse a salir.*

*El **Método base On Exit** es perfecto para:*

- *Guardar (localmente, en el disco) preferencias o parámetros a reutilizar al comienzo de la siguiente sesión en el **Método base On Startup .***
- *Realizar otras acciones automáticamente cada vez que se salga de la base.*

***Nota:** no olvide que el proceso creado por el **Método base On Exit** es un proceso local/cliente, por lo tanto no puede acceder al archivo de datos. Si el **Método base On Exit** realiza una consulta o una ordenación, un cliente 4D que está a punto de salir quedará "congelado" y en realidad no saldrá de la aplicación. Si necesita acceder a datos cuando un cliente sale de la aplicación, cree un nuevo proceso global desde el cual el **Método base On Exit** pueda acceder al archivo de datos. En este caso, asegúrese de que el nuevo proceso termine correctamente antes del final de la ejecución del **Método base On Exit** (utilizando por ejemplo variables interproceso).*
***Nota**: en un entorno cliente/servidor, **Método base On Exit** se comporta de manera diferente dependiendo de si el usuario sale manualmente (vía el comando de menú **Salir** o una llamada al comando **QUIT 4D**) o que 4D Server se cierre, lo que obliga a todos los clientes a salir.*
*Cuando se sale de 4D Server y se da un tiempo de corte (por ejemplo, 10 minutos), cada cliente conectado muestra un mensaje de advertencia y si el usuario sale durante el período de tiempo determinado, el **Método base On Exit** se ejecuta normalmente. Sin embargo, en otros casos (por ejemplo, el usuario no responde a tiempo, el servidor solicita salir inmediatamente o el administrador desconecta manualmente al cliente), el **Método base On Exit** se ejecuta al mismo tiempo que la conexión al servidor se cierra. Como resultado, el código en **Método base On Exit** no puede iniciar otro proceso local o de servidor y no puede esperar a que se cancelen otros procesos (ni estos procesos pueden seguir accediendo al servidor). Si intenta hacerlo, se genera un error de red (como 10001 o 10002) ya que la conexión al servidor ya está cerrada.*
*Para detener correctamente los procesos en ejecución en el caso de paradas inesperadas, debe probar el comando **Process aborted** en cada bucle (for, while, repeat) lo que puede durar más de un segundo. **Process aborted** devuelve **true** si 4D (local, remoto o 4D Server) está a punto de salir, lo que significa que el procesamiento está a punto de detenerse de inmediato. En este caso, cancele todos los procesos (**CANCEL TRANSACTION**, etc.) y salga lo más rápido posible. Aunque tiene tiempo si el usuario sale manualmente, no tiene tiempo si la aplicación es forzada a salir.*

Ejemplo

El siguiente ejemplo cubre todos los métodos utilizados en una base que sigue los eventos más importantes que ocurren durante una sesión de trabajo y escribe una descripción en un documento de de texto llamado "Diario."

- *El **Método base On Startup** inicializa la variable interproceso ◇◇vbQuit4D, la cual señala los procesos utilizados sin importar si se está saliendo o no de la base. También crea el archivo de diario, si no existe aún.*

```
` Método base On Startup
C_TEXT(◇vtIPMessage)
C_BOOLEAN(◇vbQuit4D)
◇vbQuit4D:=False

If(Test path name("Journal")#Is a document)
```

```
        $vhDocRef:=Create document("Journal")
        If(OK=1)
            CLOSE DOCUMENT($vhDocRef)
        End if
    End if
    WRITE JOURNAL("Apertura de sesión")
```

- *El método proyecto **WRITE JOURNAL**, utilizado como subrutina por los otros métodos, escribe la información que recibe, en el archivo historial:*

```
    ` Método proyecto WRITE JOURNAL
    ` WRITE JOURNAL ( Text )
    ` WRITE JOURNAL ( Event description )
    C_TEXT($1)
    C_TIME($vhDocRef)

    While(Semaphore("$Journal"))
        DELAY PROCESS(Current process;1)
    End while
    $vhDocRef:=Append document("Journal")
    If(OK=1)
        PROCESS PROPERTIES(Current process;$vsProcessName;$vlState;$vlElapsedTime;$vbVisible)
        SEND PACKET($vhDocRef;String(Current date)+Char(9)+String(Current time)+Char(9)
        +String(Current process)+Char(9)+$vsProcessName+Char(9)+$1+Char(13))
        CLOSE DOCUMENT($vhDocRef)
    End if
    CLEAR SEMAPHORE("$Journal")
```

Note que el documento se abre y cierra cada vez. También observe que el uso de un semáforo como "protección de acceso" al documento—no queremos dos procesos tratando de acceder el archivo diario al mismo tiempo.

- *El método proyecto **M_AGREGAR_REGISTROS** se ejecuta cuando el comando de menú **Agregar registro** se selecciona en modo Aplicación:*

```
    ` Método de proyecto M_AGREGAR_REGISTROS
SET MENU BAR(1)
Repeat
    ADD RECORD([Tabla1];*)
    If(OK=1)
        ESCRIBIR DIARIO("Adición del registro #"+String(Record number([Tabla1]))+" en Tabla1")
    End if
Until((OK=0)|◇vbQuit4D)
```

Este método efectúa un bucle hasta que el usuario cancela la entrada de datos o sale de la base.

- *El formulario entrada de la [Tabla 1] incluye el tratamiento de los eventos On Outside Call. De manera que, incluso si un proceso está en entrada de datos, puede salir sin problemas y el usuario puede guardar o no la entrada de datos actual:*

```
    ` [Table1]; Método formulario "Input"
Case of
    :(Form event=On Outside Call)
        If(◇vtIPMessage="QUIT")
            CONFIRM(""Quiere guardar las modificaciones realizadas a este registro?")
```

```
    If(OK=1)
        ACCEPT
    Else
        CANCEL
    End if
End if
End case
```

```
    `  [Tabla1];"Entrada"
Case of
    :(Form event=On Outside Call)
        If(◇vtIPMessage="QUIT")
            CONFIRM(
            If(OK=1)
```

```
            ACCEPT
        Else
            CANCEL
        End if
    End if
End case
```

- *El método proyecto M_QUIT se ejecuta cuando el comando **Salir** del menú **Archivo** se selecciona en el entorno Aplicación:*

```
` Método proyecto M_QUIT
$vlProcessID:=New process("SALIR";32*1024;"$SALIR")
```

*Este método utiliza un truco. Cuando se llama a **QUIT 4D**, el comando tiene un efecto inmediato. Por lo tanto, el proceso del cual se hace la llamada está en "modo detención" hasta que la base se cierre en realidad. Como este proceso puede ser uno de los procesos en el cual ocurre la entrada da datos, la llamada a **QUIT 4D** se realiza en un proceso local que empieza sólo con este propósito. Este es el método **SALIR:***

```
` Método de proyecto SALIR
CONFIRM("¿Está seguro de que quiere salir?")
If(OK=1)
    ESCRIBIR DIARIO("Salida de la base")
    QUIT 4D
    ` QUIT 4D tiene un efecto inmediato, ninguna línea de código a continuación se ejecutará
    ` ...
End if
```

- *Finalmente, he aquí el **Método base On Exit** el cual indica a todos los procesos de usuario abiertos que "¡Es hora de salir!". La variable <>vbQuit4D toma el valor True y envía mensajes interproceso a los procesos de usuario que están realizando entrada de datos:*

```
` Método de base On Exit
◇vbQuit4D:=True
Repeat
    $vbHecho:=True
    For($vlProceso;1;Count tasks)
        PROCESS PROPERTIES($vlProceso;$vsNombreProceso;$vlEstado;$vlTiempo;$vbVisible)
        If((((($vsNombreProceso="ML_@")|($vsNombreProceso="M_@")))&($vlEstado>=0))
            $vbHecho:=False
            ◇vtlPMensaje:="SALIR"
            BRING TO FRONT($vlProceso)
            POST OUTSIDE CALL($vlProceso)
            $vhlnicio:=Current time
            Repeat
                DELAY PROCESS(Current process;60)
            Until((Process state($vlProceso)<0)|((Current time-$vhlnicio)>=?00:01:00?))
        End if
    End for
Until($vbHecho)
ESCRIBIR DIARIO("Cierre de sesión")
```

***Nota**: Los procesos que comienzan con "ML_..." o "M_..." son iniciados por comandos de menú para los cuales la propiedad **Nuevo proceso** ha sido seleccionada. En este ejemplo, estos son los procesos iniciados cuando se selecciona el comando de menú **Agregar registro**.*

La prueba (Current time-$vhInicio)>=?00:01:00? permite al método de base salir del bucle "en espera de otro proceso" si el otro proceso no actúa de inmediato.

El siguiente es un ejemplo típico de un archivo de diario producido por la base:

2/6/03 15:47:25 1Apertura de sesión proceso principal
2/6/03 15:55:43 5 ML_1Adición del registro #23 en Tabla1
2/6/03 5:55:46 5 ML_1Adición del registro #24 en Tabla1
2/6/03 15:55:54 6 $DO_QUITSalir de la base
2/6/03 15:55:58 7 $xxCerrar sesión

***Nota**: $xx es el nombre del proceso local iniciado por 4D para ejecutar el **Método base On Exit**.*

Método base On Host Database Event

$1 -> Método base On Host Database Event

Parámetro	Tipo		Descripción
$1	Entero largo	⇐	Código del evento

Description

*El **Método base On Host Database Event** permite a los componentes 4D ejecutar código cuando se abre y cierra la base local.*

***Nota**: por razones de seguridad, la ejecución de este método base debe ser autorizada explícitamente en la base local. Para obtener más información sobre este punto, consulte el manual de Diseño.*

*El **Método base On Host Database Event** se ejecuta automáticamente solamente en bases utilizadas como componentes de bases locales (cuando se autoriza en las propiedades de la base local). Se llama cuando se producen eventos relacionados con la apertura y cierre de la base local.*

*Para procesar un evento, debe probar el valor del parámetro $1 en el método, y compararlo con una de las siguientes constantes, disponibles en el tema "**Eventos de la base**":*

Constante	Tipo	Valor	Comentario
On after host database exit	Entero largo	4	El **Semaphore** de la base local acaba de terminar su ejecución
On after host database startup	Entero largo	2	El **Método base On Startup** de la base local acaba de terminar su ejecución
On before host database exit	Entero largo	3	La base local se está cerrando. El **Semaphore** de la base local aún no se ha llamado. El **Semaphore** de la base loal no se llama mientras el **Método base On Host Database Event** del componente se esté ejecutando
On before host database startup	Entero largo	1	La base local se ha iniciado. El **Método base On Startup** de la base local aún no se ha llamado. El método base On Startup de la base local no se llama mientras el **Método base On Host Database Event** del componente se esté ejecutando

Esto permite a los componentes 4D cargar y guardar preferencias o estados de usuario relacionados con el funcionamiento de la base local.

Ejemplo

Ejemplo de estructura tipo de un método base On Host Database Event:

```
// Método base On Host Database Event
C_LONGINT($1)
Case of
   :($1=On before host database startup)
// poner aquí el código a ejecutar antes del método base "On Startup"
// de la base local
   :($1=On after host database startup)
// poner aquí el código a ejecutar después del método base "On Startup"
// de la base local
   :($1=On before host database exit)
// poner aquí el código a ejecutar antes del método base "On Exit"
// de la base local
   :($1=On after host database exit)
// poner aquí el código a ejecutar después del método base "On Exit"
// de la base local
End case
```

Método base On Server Close Connection

$1, $2, $3 -> Método base On Server Close Connection

Parámetro	Tipo		Descripción
$1	Entero largo	⬅	Número de usuario utilizado internamente por 4D Server para identificar los usuarios
$2	Entero largo	⬅	Número de conexión utilizada internamente por 4D Server para identificar una conexión
$3	Entero largo	⬅	Obsoleto: devuelve siempre 0 pero debe declararse

Descripción

*El **Método base On Server Close Connection** se llama en el equipo servidor cada vez que termina un proceso 4D Client.*

*Como para el **Método base On Server Open Connection**, 4D Server pasa tres parámetros de tipo entero largo al **Método base On Server Close Connection**. Por otra parte, 4D Server no espera un resultado en retorno.*

El método debe contener la declaración explícita de tres parámetros Entero largo:

```
C_LONGINT($1;$2;$3)
```

Esta tabla detalla la información ofrecida por los tres parámetros pasados al método base:

Parámetro	Descripción
$1	Número de usuario utilizado internamente por 4D Server para identificar usuarios
$2	Número de conexión utilizado internamente por 4D Server para identificar una conexiónn
$3	Obsoleto: devuelve siempre 0 pero debe declararse

*El **Método base On Server Close Connection** es el inverso exacto del **Método base On Server Open Connection**. Para mayor información y una descripción de este método base, así como para la descripción de los **procesos 4D Client**, ver la descripción de este método base.*

Ejemplo

*Ver el primer ejemplo para **Método base On Server Open Connection**.*

Método base On Server Open Connection

$1, $2, $3 -> Método base On Server Open Connection -> $0

Parámetro	Tipo		Descripción
$1	Entero largo	⬅	Número de usuario utilizado internamente por 4D Server para identificar los usuarios
$2	Entero largo	⬅	Número de conexión utilizado internamente por 4D Server para identificar una conexión
$3	Entero largo	⬅	Obsoleto: devuelve siempre 0 (pero debe declararse)
$0	Entero largo	➲	0 o se omite = conexión aceptada; otro valor = conexión rechazada

¿Cuándo se llama el método base On Server Open Connection?

El **Método base On Server Open Connection** se llama una vez en el equipo servidor cada vez que un equipo 4D remoto inicia un proceso de conexión. El **Método base On Server Open Connection** NO se invoca por otro entorno 4D diferente de 4D Server.

El **Método base On Server Open Connection** se llama cada vez que:

- un 4D remoto se conecta (inicio del proceso principal)
- un 4D remoto abre el entorno Diseño (inicio del proceso de Diseño)
- un 4D remoto inicia un proceso global, (cuyo nombre o comienza por "$") lo cual necesita de la creación de un proceso cooperativo en el servidor **(*)**. Este proceso puede crearse utilizando el comando **New process**, un comando de menú o la caja de diálogo "Ejecutar un método".

En cada caso con un 4D remoto, se inician tres procesos. Uno en la máquina cliente y otros dos en el equipo servidor. En la máquina cliente, el proceso ejecuta el código y envía las peticiones a 4D Server. En el equipo servidor, el **proceso 4D Client** mantiene el entorno de la base de datos del proceso cliente (las selecciones actuales y el bloqueo de registros para el proceso usuario) y responde a las peticiones enviadas por el proceso ejecutado en la máquina cliente. El **proceso base 4D Client** está a cargo de controlar el proceso 4D Client correspondiente.

(*) A partir de 4D v13, por razones de optimización los procesos servidores (proceso apropiativo para los accesos al motor de la base y proceso cooperativo para el acceso al lenguaje) sólo se crean durante la ejecución del código del lado del cliente. Por ejemplo, estos son los detalles de una secuencia de código 4D que se ejecuta en un nuevo proceso cliente:

```
// el proceso global comienza sin un nuevo proceso en el servidor, como un proceso local.
CREATE RECORD([Table_1])
[Table_1]field1_1:="Hello world"
SAVE RECORD([Table_1]) // creación aquí del proceso apropiativo en el servidor
$serverTime:=Current time(*) // creación aquí del proceso cooperativo en el servidor
// llamada de On Server Open Connection
```

Importante: las conexiones web y las conexiones SQL no provocan la ejecución del **Método base On Server Open Connection**. Cuando un navegador web se conecta a 4D Server, se llaman el **Método de base On Web Authentication** (si lo hay) y/o la **Método base On Web Connection**.
Cuando 4D Server recibe una petición SQL, se llama el **Método base On SQL Authentication**(si existe). Para mayor información, consulte la descripción de este método base en el manual de Lenguaje 4D.

Importante: cuando se inicia un procedimiento almacenado, el **Método base On Server Open Connection** NO se llama. Los **Procedimientos almacenados** son procesos servidor y no procesos 4D Client. Ellos ejecutan el código en el equipo servidor, pero no responden a las peticiones intercambiadas por 4D client (u otros clientes) y 4D Server**.**

¿Cómo se llama al método base On Server Open Connection?

El **Método base On Server Open Connection** se ejecuta en el equipo servidor en el proceso 4D Client que provocó la llamada del método.

Por ejemplo, si un 4D remoto se conecta a una base 4D Server interpretada, se inicia el proceso usuario, el proceso de diseño y el proceso de registro del cliente (por defecto). El **Método base On Server Open Connection** se ejecuta tres veces seguidas. La primera vez dentro del proceso principal, la segunda vez en el proceso de inscripción del cliente y la tercera vez en el proceso de diseño. Si los tres procesos son respectivamente el sexto, séptimo y octavo proceso a iniciar en el equipo servidor, y si llama **Current process** desde el **Método base On Server Open Connection**, la primera vez **Current process** devuelve 6, la segunda vez 7 y la tercera 8.

Note que el **Método base On Server Open Connection** se ejecuta en el equipo servidor, al interior del proceso 4D Client en el servidor, independiente del proceso ejecutado en el cliente. Adicionalmente, en el momento en que se invoca el método, el proceso 4D Client no se ha nombrado aún (**PROCESS PROPERTIES** no devolverá en este momento el nombre del proceso 4D Client).

El **Método base On Server Open Connection** no tiene acceso a la tabla de las variables proceso del proceso ejecutado en el client. Esta tabla reside en el equipo client, no en el equipo servidor.

Cuando el **Método base On Server Open Connection** accede a una variable proceso, trabaja con una tabla de variables proceso particular, creada dinámicamente por el proceso 4D Client.

4D Server pasa tres parámetros de tipo Entero largo al **Método base On Server Open Connection** y espera un resultado Entero largo. El método debe por lo tanto ser declarado explícitamente con tres parámetros Entero largo así como también con un resultado de función Entero largo:

```
C_LONGINT($0;$1;$2;$3)
```

Si no devuelve un valor en $0, por consiguiente deja la variable indefinida o inicializada en cero, 4D Server estima que el método base acepta la conexión. Si no acepta la conexión, devuelve un valor no nulo en $0.

Esta tabla detalla la información ofrecida por los tres parámetros pasados en el método base:

Parámetro	Descripción
$1	Número de usuario utilizado internamente por 4D Server para identificar los usuarios
$2	Número de conexión utilizado internamente por 4D Server para identificar una conexión
$3	Obsoleto: siempre devuelve 0 pero debe declararse

*Estos números de referencia no son utilizables directamente como fuentes de información a pasar, por ejemplo, como parámetros a un comando 4D. Sin embargo, ofrecen una manera única de identificar un proceso 4D Client entre el **Método base On Server Open Connection** y el **Método base On Server Close Connection**. La combinación de estos valores es única en cualquier momento de una sesión 4D Server. Al guardar esta información en una tabla o en un array interproceso, los dos métodos base pueden intercambiar información. En el ejemplo al final de esta sección, los dos métodos base utilizan esta información para almacenar la fecha y hora de inicio y fin de una conexión en el mismo registro de una tabla.*

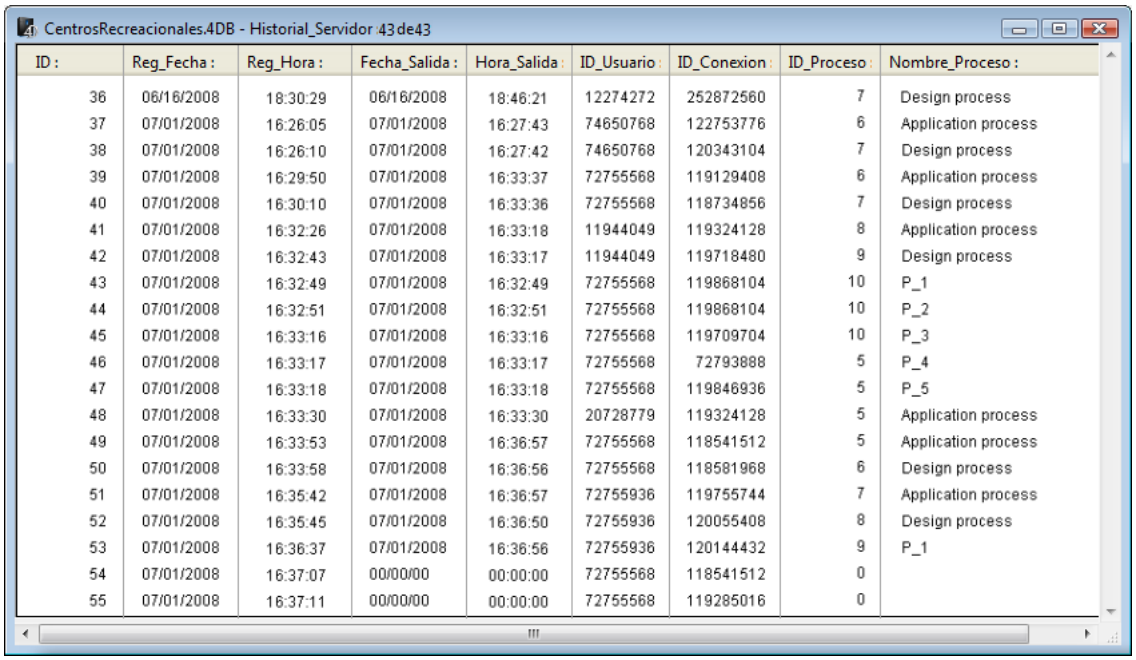
Ejemplo 1

*El siguiente ejemplo muestra cómo mantener un historial de las conexiones a la base de datos utilizando el **Método base On Server Open Connection** y utilizando el **Método base On Server Close Connection**. La tabla [Server Log] (mostrada a continuación) se utiliza para hacer seguimiento a los procesos de conexión:*

*La información almacenada en esta tabla es administrada por el **Método base On Server Open Connection** y el **Método base On Server Close Connection** listado a continuación:*

```
  ` Método base On Server Open Connection

C_LONGINT($0;$1;$2;$3)
  ` Crear un registro [Server Log]
CREATE RECORD([Server Log])
[Server Log]Log ID:=Sequence number([Server Log])
  ` Guardar el historial Fecha y Hora
[Server Log]Log Date:=Current date
[Server Log]Log Time:=Current time
  ` Guarda la información de conexión
[Server Log]User ID:=$1
[Server Log]Connection ID:=$2
SAVE RECORD([Server Log])
  ` No devuelve error de manera que la conexión puede continuar
$0:=0

  ` Método base On Server Close Connection
C_LONGINT($1;$2;$3)
  ` Recuperar el registro [Server Log]
QUERY([Server Log];[Server Log]User ID=$1;*)
QUERY([Server Log];&;[Server Log]Connection ID=$2;*)
QUERY([Server Log];&;[Server Log]Process ID=0)
  ` Guardar fecha y hora de desconexión
[Server Log]Exit Date:=Current date
[Server Log]Exit Time:=Current time
  ` Guardar información proceso
[Server Log]Process ID:=Current process
PROCESS PROPERTIES([Server Log]Process ID;$vsProcName;$vlProcState;$vlProcTime)
[Server Log]Process Name:=$vsProcName
SAVE RECORD([Server Log])
```

Estas son algunas entradas en [Server Log] mostrando varias conexiones remotas:



CentrosRecreacionales.4DB - Historial_Servidor 43 de 43

ID :	Reg_Fecha :	Reg_Hora :	Fecha_Salida :	Hora_Salida :	ID_Usuario :	ID_Conexion :	ID_Proceso :	Nombre_Proceso :
36	06/16/2008	18:30:29	06/16/2008	18:46:21	12274272	252872560	7	Design process
37	07/01/2008	16:26:05	07/01/2008	16:27:43	74650768	122753776	6	Application process
38	07/01/2008	16:26:10	07/01/2008	16:27:42	74650768	120343104	7	Design process
39	07/01/2008	16:29:50	07/01/2008	16:33:37	72755568	119129408	6	Application process
40	07/01/2008	16:30:10	07/01/2008	16:33:36	72755568	118734856	7	Design process
41	07/01/2008	16:32:26	07/01/2008	16:33:18	11944049	119324128	8	Application process
42	07/01/2008	16:32:43	07/01/2008	16:33:17	11944049	119718480	9	Design process
43	07/01/2008	16:32:49	07/01/2008	16:32:49	72755568	119868104	10	P_1
44	07/01/2008	16:32:51	07/01/2008	16:32:51	72755568	119868104	10	P_2
45	07/01/2008	16:33:16	07/01/2008	16:33:16	72755568	119709704	10	P_3
46	07/01/2008	16:33:17	07/01/2008	16:33:17	72755568	72793888	5	P_4
47	07/01/2008	16:33:18	07/01/2008	16:33:18	72755568	119846936	5	P_5
48	07/01/2008	16:33:30	07/01/2008	16:33:30	20728779	119324128	5	Application process
49	07/01/2008	16:33:53	07/01/2008	16:36:57	72755568	118541512	5	Application process
50	07/01/2008	16:33:58	07/01/2008	16:36:56	72755568	118581968	6	Design process
51	07/01/2008	16:35:42	07/01/2008	16:36:57	72755936	119755744	7	Application process
52	07/01/2008	16:35:45	07/01/2008	16:36:50	72755936	120055408	8	Design process
53	07/01/2008	16:36:37	07/01/2008	16:36:56	72755936	120144432	9	P_1
54	07/01/2008	16:37:07	00/00/00	00:00:00	72755568	118541512	0	
55	07/01/2008	16:37:11	00/00/00	00:00:00	72755568	119285016	0	

Ejemplo 2

El siguiente ejemplo evita una nueva conexión entre las 2 y 4 A.M.

```
  ` Método base On Server Open Connection
C_LONGINT($0;$1;$2;$3)

If((?02:00:00?<=Current time)&(Current time<?04:00:00?))
   $0:=22000
Else
   $0:=0
End if
```

Método base On Server Shutdown

Método base On Server Shutdown
Este comando no requiere parámetros

---

*El **Método base On Server Shutdown** se llama una vez en el equipo servidor cuando la base actual se cierra en 4D Server. El **Método base On Server Shutdown** NO es llamado por otro entorno 4D diferente de 4D Server.*

*Para cerrar la base actual en el servidor, puede seleccionar el comando de menú **Cerrar la base...** en el servidor. También puede elegir el comando **Salir** o llamar al comando **QUIT 4D** dentro de un procedimiento almacenado ejecutado en el servidor.*

Cuando se inicia el proceso de cierre de la base, 4D efectúa las siguientes acciones:

- *Si no hay un **Método base On Server Shutdown**, 4D Server aborta cada proceso en ejecución uno por uno, sin distinción.*
- *Si existe un **Método base On Server Shutdown**, 4D Server ejecuta este método en un nuevo proceso local. Por lo tanto puede utilizar este método base para informar los otros procesos, vía la comunicación interproceso, que deben detener su ejecución. Note que 4D Server saldrá finalmente, el **Método base On Server Shutdown** puede efectuar todas las operaciones de limpieza o cierre que usted quiera, pero no puede rehusarse a salir y en algún momento terminará.*

*El **Método base On Server Shutdown** es el lugar ideal para:*

- *Detener los procedimientos almacenados lanzados automáticamente cuando se abre la base.*
- *Guardar (localmente, en disco) las preferencias o los parámetros a reutilizar al inicio de la sesión siguiente en el **Método base On Server Startup**.*
- *Efectuar cualquier otra acción que quiera activar automáticamente cada vez que salga de la base.*

***Importante:** si utiliza el **Método base On Server Shutdown** para cerrar los procedimientos almacenados, recuerde que el servidor sale una vez se ejecuta el **Método base On Server Shutdown** (y no los procedimientos almacenados). Si los procedimientos almacenados aún están corriendo en este momento, se abortarán. Por lo tanto, si quiere asegurarse de que los procedimientos almacenados se ejecuten completamente antes de ser abortados por el servidor, el **Método base On Server Shutdown** debe indicar a los procedimientos almacenados que deben terminar su ejecución (por ejemplo, utilizando una variable interproceso) y debe permitirles cerrar (por medio de un bucle de x segundos u otra variable interproceso).*

*Si quiere que el código se ejecute automáticamente en un equipo cliente cuando un 4D remoto deja de conectarse al servidor, utilice el **Semaphore**.*

Método base On Server Startup

Método base On Server Startup
Este comando no requiere parámetros

---

*El **Método base On Server Startup** se llama una vez en el equipo servidor cuando abre una base con 4D Server. El **Método base On Server Startup** NO se ejecuta en un entorno diferente a 4D Server.*

*El **Método base On Server Startup** es la ubicación ideal para:*

- *Inicializar las variables interproceso utilizadas durante toda la sesión 4D Server.*
- *Iniciar automáticamente los **Procedimientos almacenados** al abrir la base.*
- *Cargar preferencias o parámetros guardados durante la sesión anterior de 4D Server.*
- *Evitar la apertura de la base si no se cumple una condición (ausencia de recursos sistema) para una llamada explícita a **QUIT 4D**.*
- *Realizar otras acciones que quiera efectuar automáticamente cada vez que se abra la base.*

*Para ejecutar código automáticamente en un equipo cliente cuando un 4D remoto se conecta al servidor, utilice el **Método base On Server Startup**.*

***Nota**: el **Método base On Server Startup** se ejecuta de manera atómica, lo que significa que ningún 4D remoto puede conectarse mientras la ejecución del método no haya terminado.*

Método base On SQL Authentication

$1, $2, $3 -> Método base On SQL Authentication -> Resultado

Parámetro	Tipo		Descripción
$1	Texto	⇐	Nombre de usuario
$2	Texto	⇐	Contraseña
$3	Texto	⇐	(Opcional) Dirección IP del cliente al origen de la petición
Resultado	Booleano	⇒	True = petición aceptada, False = petición rechazada

---

*El **Método base On SQL Authentication** puede utilizarse para filtrar las peticiones enviadas al servidor SQL integrado de 4D. Este filtro puede estar basado en el nombre y contraseña como también de manera opcional en la dirección IP del usuario. El desarrollador puede utilizar su propia tabla de usuarios o la de los usuarios 4D para evaluar los identificadores de conexión. Una vez validada la conexión, el comando **CHANGE CURRENT USER** puede utilizarse para controlar el acceso de las peticiones dentro de la base 4D.*

*Cuando existe, el **Método base On SQL Authentication** es llamado automáticamente por 4D o 4D Server en cada conexión externa al servidor SQL. Por lo tanto el sistema interno de gestión de los usuarios de 4D no está activado. La conexión se acepta sólo si el método de base devuelve **True** en $0 y si el comando **CHANGE CURRENT USER** se ha ejecutado con éxito. Si una de estas condiciones no se cumple, la petición se rechaza.*

***Nota**: la instrucción **SQL LOGIN(SQL_INTERNAL**;$usuario;$contraseña) no llama al **Método base On SQL Authentication** ya que es una conexión interna en este caso.*

El método de base recibe hasta tres parámetros de tipo Texto, pasados por 4D ($1, $2 y $3) y devuelve un booleano, $0. Esta es la descripción de estos parámetros:

Parámetros	Tipo	Descripción
$1	Texto	Nombre de usuario
$2	Texto	Contraseña
$3	Texto	(opcional) Dirección IP del cliente al origen de la petición (*)
$0	Booleano	True = petición aceptada, False = petición rechazada

*(*) 4D devuelve las direcciones IPv4 en un formato híbrido IPv6/IPv4 escrito con un prefijo de 96 bits, por ejemplo ::ffff:192.168.2.34 para la dirección IPv4 192.168.2.34. Para mayor información, consulte la sección **Soporte de IP v6**.*

Debe declarar estos parámetros de esta forma:

```
` Método de base On Web Authentication

C_TEXT($1;$2;$3;$4)
C_BOOLEAN($0)
` Código para el método
```

La contraseña ($2) se recibe como texto estándar.

*Debe controlar los identificadores de la conexión SQL en el **Método base On SQL Authentication**. Por ejemplo, puede verificar el nombre y la contraseña utilizando una tabla de usuarios personalizada. Si los identificadores son válidos, pase **True** en $0 para aceptar la conexión y la petición. 4D abre una sesión SQL para el usuario. De lo contrario, pase **False** en $0; en este caso, la conexión se rechaza.*

***Nota:** si el **Método base On SQL Authentication** no existe, la conexión se evalúa utilizando el sistema integrado de gestión de usuarios de 4D (si está activo, en otras palabras, si una contraseña ha sido asignada al Diseñador). Si este sistema no está activado, los usuarios están conectados con los derechos de acceso del Diseñador (acceso libre).*

*Si pasa **True** en $0, debe llamar exitosamente al comando **CHANGE CURRENT USER** en el **Método base On SQL Authentication** para que la petición sea aceptada y para que 4D abra una sesión SQL para el usuario.*

El uso de este comando se recomienda porque permite un mayor nivel de seguridad. Esta autenticación virtual tiene la doble ventaja de permitir el control de las acciones de conexión y de ocultar para el exterior los identificadores de la conexión en la sesión SQL 4D.

*Cuando el sistema de contraseñas integrado de 4D no está activo, la ejecución del comando **CHANGE CURRENT USER** no tiene efecto; los usuarios se conectan con los derechos de acceso del Diseñador.*

*Este ejemplo del **Método base On SQL Authentication** verifica que la petición de conexión provenga de la red interna, valida los identificadores y luego asigna los derechos de accesos "sql_user" para la sesión SQL.*

```
C_TEXT($1;$2;$3;$4)
C_BOOLEAN($0)
  `$1: usuario
  `$2: contraseña
  `{$3: dirección IP del cliente}
ON ERR CALL("SQL_error")
If(DirIPInterna($3))
  `El método DirIPInterna verifica si la dirección IP es interna
   If($1="victor") & ($2="hugo")
      CHANGE CURRENT USER("sql_user";"")
      If(OK=1)
         $0:=True
```

```
            Else
                $0:=False
            End if
        Else
            $0:=False
        End if
    Else
        $0:=False
    End if
```

Método base On Startup

Método base On Startup
Este comando no requiere parámetros

---

*El **Método base On Startup** se ejecuta una sola vez, al momento de la apertura de la base.*
Esto ocurre en los siguientes entornos 4D:

- *4D en modo local*
- *4D en modo remoto (del lado del cliente, una vez la conexión haya sido aceptada por 4D Server)*
- *Aplicación 4D compilada y fusionada con 4D Volume Desktop*

***Nota**: el **Método base On Startup** NO es ejecutado por 4D Server.*
*El **Método base On Startup** es invocado automáticamente por 4D; a diferencia de los métodos proyecto, usted no puede llamar este método base por programación. Sin embargo, puede ejecutarlo desde el editor de métodos. También puede utilizar subrutinas.*
*El **Método base On Startup** es perfecto para:*

- *Inicializar variables interproceso que utilizará durante toda la sesión de trabajo.*
- *Iniciar procesos automáticamente cuando abre una base.*
- *Cargar preferencias o parámetros guardados durante la sesión de trabajo anterior.*
- *Evitar la apertura de la base si no se cumple una condición (por ejemplo, si falta un recurso del sistema) llamando explícitamente **QUIT 4D**.*
- *Realizar otras acciones que quiera ejecutar automáticamente cada vez que abra una base.*

*Sin embargo, le recomendamos NO lanzar trabajos de impresión desde el **Método base On Startup** .*

Ejemplo

Vea el ejemplo en la sección .

Método base On System Event

$1 -> Método base On System Event

Parámetro	Tipo		Descripción
$1	Entero largo	⇐	Código del evento

Descripción

*El **Método base On System Event** se llama cada vez que ocurre un evento sistema. Esto concierne a todos los entornos 4D: 4D (todos los modos) y 4D Server, así como también las aplicaciones 4D compiladas y fusionadas con 4D Volume Desktop.*

*Para procesar un evento, debe probar el valor del parámetro $1 al interior del método y compararlo con una de las siguientes constantes, del tema **Eventos de la base**:*

Constante	Tipo	Valor	Comentario
On application background move	Entero largo	1	La aplicación 4D pasa al fondo
On application foreground move	Entero largo	2	La aplicación 4D pasa al primer plano

Estos eventos se generan cuando la aplicación 4D cambia de nivel, sin importar la acción del usuario que genera este cambio. Por ejemplo:

- *clic en la ventana de la aplicación o de otra aplicación,*
- *selección utilizando el atajo de teclado **Alt+Tab** (Windows) o **Comando+Tab** (Mac OS),*
- *selección del comando **Ocultar** en el dock (Mac OS),*
- *clic en el icono de la aplicación en el dock o la barra de tareas,*
- *clic en el botón de minimización de la ventana principal (Windows).*

Es absolutamente necesario declarar el parámetro $1 (entero largo) en el método base. La estructura del código del método base será entonces:

```
// Método base On System Event

C_LONGINT($1)
Case of
   :($1=On application background move)
 //Hacer algo
   :($1=On application foreground move)
 //Hacer otra cosa
End case
```

Método base On Web Authentication

$1, $2, $3, $4, $5, $6 -> Método base On Web Authentication -> Resultado

Parámetro	Tipo		Descripción
$1	Texto	⇐	URL
$2	Texto	⇐	Encabezado HTTP + Cuerpo HTTP
$3	Texto	⇐	Dirección IP del navegador
$4	Texto	⇐	Dirección IP del servidor
$5	Texto	⇐	Nombre de usuario
$6	Texto	⇐	Contraseña
Resultado	Booleano	⇒	True = petición aceptada, False = petición rechazada

Descripción

El **Método base On Web Authentication** está a cargo de administrar el acceso al motor del servidor web. Es llamado automáticamente por 4D o 4D Server cuando una petición de un navegador web requiere la ejecución de un método 4D en el servidor (llamada de un método vía un URL 4DACTION o una etiqueta 4DSCRIPT, etc.).

Este método recibe seis parámetros de tipo Texto, pasados por 4D: $1, $2, $3, $4, $5, y $6 y devuelve un booleano, $0. La descripción de estos parámetros es la siguiente:

Parámetros	Tipo	Descripción
$1	Texto	URL
$2	Texto	Encabezado + Cuerpo HTTP (32 KB máximo)
$3	Texto	Dirección IP del navegador
$4	Texto	Dirección IP que llama al servidor
$5	Texto	Nombre del usuario
$6	Texto	Contraseña
$0	Booleano	True = petición aceptada, False = petición rechazada

Debe declarar estos parámetros de esta forma:

```
` Método de base On Web Authentication

C_TEXT($1;$2;$3;$4;$5;$6)
C_BOOLEAN($0)

` Código para el método
```

Nota: todos los parámetros del **Método base On Web Authentication** no se llenarán. La información recibida por el método de base depende las opciones que haya seleccionado previamente en la caja de diálogo de Propiedades de la base. Consulte la sección **Seguridad de las conexiones**).

- **URL**

El primer parámetro ($1) es el URL introducido por el usuario en el área ubicación de su navegador web, menos la dirección local.

Tomemos el ejemplo de una conexión de Intranet. Supongamos que la dirección IP de su equipo servidor web 4D es 123.4.567.89. La siguiente tabla muestra los valores de $1 dependiendo del URL introducido en el navegador web:

URL introducido en el navegador web	Valor del parámetro $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clientes	/Clientes
http://123.4.567.89/Clientes	/Clientes
http://123.4.567.89/Clientes/Añadir	/Clientes/Añadir
123.4.567.89/Hacer_esto/Si_OK/Hacer_eso	/Hacer_esto/Si_OK/Hacer_eso

- **Encabezado y cuerpo de la petición HTTP**

El segundo parámetro ($2) es el encabezado y el cuerpo de la petición HTTP enviada por el navegador web. Note que esta información se pasa al **Método base On Web Authentication** tal como está. El contenido varía en función del tipo de navegador web que esté intentando la conexión. Si su aplicación manipula esta información, es su decisión si analiza el encabezado y el cuerpo.

Notas:

- Por razones de rendimiento, el tamaño de los datos que transita vía el parámetro $2 no debe superar los 32 KB. De lo contrario serán truncados por el servidor HTTP de 4D.
- Para mayor información sobre este parámetro, consulte la descripción del **Método base On Web Connection**.
- **Dirección IP del navegador**
  El tercer parámetro $3 recibe la dirección IP del equipo navegador. Esta información permite distinguir entre las conexiones de Intranet e Internet.
  Nota: 4D devuelve las direcciones IPv4 en un formato híbrido IPv6 escritos con un prefijo de 96 bits, por ejemplo ::ffff:192.168.2.34 para la dirección IPv4 192.168.2.34. Para mayor información, consulte la sección **Soporte de IP v6**.

- ***Dirección IP para llamar al servidor***
  *El cuarto parámetro $4 recibe la dirección IP utilizada para llamar al servidor Web. 4D a partir de la versión 6.5 autoriza el multi-homing, permitiendo explotar equipos con más de una dirección IP. Para mayor información, consulte la sección **QR DELETE COLUMN**.*
- ***Nombre del usuario y contraseña***
  *Los parámetros $5 y $6 reciben el nombre de usuario y contraseña introducidos por el usuario en la caja de diálogo estándar de identificación mostrada por el navegador. Esta caja de diálogo aparece para cada conexión, si una opción de gestión de contraseñas ha sido seleccionada en las Propiedades de la base (ver la sección **Seguridad de las conexiones**).*

***Nota:** si el nombre de usuario enviado por el navegador existe en 4D, el parámetro $6 (la contraseña del usuario) no se devuelve por razones de seguridad.*

● ***Parámetro $0***

*El **Método base On Web Authentication** devuelve un booleano en $0:*

- *Si $0 es **True**, la conexión es aceptada.*
- *Si $0 es **False**, la conexión es rechazada.*

*El **Método base On Web Connection** sólo se ejecuta si la conexión ha sido aceptada por **On Web Authentication**.*

***Advertencia:** si no se pasa ningún valor en $0 o si $0 no se define en el **Método base On Web Authentication**, la conexión se considerará como aceptada y se ejecuta el **Método base On Web ConnectionIs Windows**.*

Notas:

- *No llame elementos de interfaz en el **Método base On Web Authentication** (**ALERT**, **DIALOG**, etc.) porque de lo contrario su ejecución se interrumpirá y la conexión será rechazada. Lo mismo sucede si se presenta un error durante su proceso.*
- *Es posible evitar la ejecución por 4DACTION o 4DSCRIPT de cada método de proyecto con la ayuda de la opción "Disponible vía las etiquetas HTML y URLs (4DACTION...) en la caja de diálogo de las Propiedades de los métodos. Para mayor información sobre este punto, consulte la sección **Seguridad de las conexiones**.*

Llamadas del método base On Web Authentication

*El **Método base On Web Authentication** se llama automáticamente, sin importar el modo, cuando una petición o proceso requiere la ejecución de un método 4D. También se llama cuando el servidor web recibe un URL estático inválido (por ejemplo, si la página estática solicitada no existe).*

*Por lo tanto el **Método base On Web Authentication** se llama en los siguientes casos:*

- *cuando 4D recibe un URL que comienza por 4DACTION/*
- *cuando 4D recibe un URL que comienza por 4DCGI/*
- *cuando 4D recibe un URL que comienza por 4DSYNC/*
- *cuando 4D recibe un URL solicitando una página estática que no existe*
- *cuando 4D recibe un URL de acceso a la raíz y no se ha definido una página de inicio en las propiedades de la base o por medio del comando **WEB SET HOME PAGE***
- *cuando 4D procesa una etiqueta 4DSCRIPT en una página semidinámica*
- *cuando 4D procesa una etiqueta 4DLOOP basada en un método en una página semidinámica.*

***Nota de compatibilidad:** el método base también se llama cuando 4D recibe un URL que comienza por 4DMETHOD/. Este URL es obsoleto y sólo se conserva por razones de compatibilidad.*

*Note que el **Método base On Web Authentication** NO se llama cuando el servidor recibe un URL solicitando una página estática válida.*

Ejemplo 1

*Ejemplo del **Método de base On Web Authentication** en modo BASIC:*

```
  `Método de base On Web Authentication
C_TEXT($5;$6;$3;$4)
C_TEXT($usuario;$contraseña;$IPNavegador;$IPServidor)
C_BOOLEAN($4Dusuario)
ARRAY TEXT($usuarios;0)
ARRAY LONGINT($nums;0)
C_LONGINT($upos)
C_BOOLEAN($0)

$0:=False

$usuario:=$5
$contraseña:=$6
$IPNavegador:=$3
$IPServidor:=$4

  `Por razones de seguridad, rechazar nombres que contengan @
If(WithWildcard($usuario)|WithWildcard($contraseña))
   $0:=False
  `El método WithWildcard se describe a continuación
Else
```

```
  `Verificar para ver si es un usuario 4D
    GET USER LIST($usuarios;$nums)
    $upos:=Find in array($usuarios;$usuario)
    If($upos >0)
        $usuario4D:=Not(Is user deleted($nums{$upos}))
    Else
        $usuario4D:=False
    End if

    If(Not($usuario4D))
  `No es un usuario definido en 4D, buscar en la tabla de usuarios Web
        QUERY([UsuariosWeb];[UsuariosWeb]usuario=$usuario;*)
        QUERY([UsuariosWeb]; & [UsuariosWeb]contraseña=$contraseña)
        $0:=(Records in selection([UsuariosWeb])=1)
    Else
        $0:=True
    End if
End if
  `¿Esta es una conexión de intranet?
If(Substring($IPNavegador;1;7)#"192.100.")
    $0:=False
End if
```

Ejemplo 2

*Ejemplo del **Método de base On Web Authentication** en modo DIGEST:*

```
  `Método de base On Web Authentication
C_TEXT($1;$2;$5;$6;$3;$4)
C_TEXT($usuario)
C_BOOLEAN($0)
$0:=False
$usuario:=$5
  `Por razones de seguridad, rechazar los nombres que contengan @
If(WithWildcard($usuario))
    $0:=False
  `El método WithWildcard se describe a continuación
Else
    QUERY([UsuariosWeb];[UsuariosWeb]usuario=$usuario)
    If(OK=1)
        $0:=Validate Digest Web Password($usuario;[UsuariosWeb]contraseña)
    Else
        $0:=False  `Usuario inexistente
    End if
End if
```

El método de proyecto WithWildcard es el siguiente:

```
  `Método WithWildcard
  `WithWildcard ( Cadena) -> Booleano
  `WithWildcard ( Nombre ) -> Contiene un carácter arroba

C_INTEGER($i)
C_BOOLEAN($0)
C_TEXT($1)

$0:=False
For($i;1;Length($1))
    If(Character code(Substring($1;$i;1))=Character code("@"))
        $0:=True
    End if
End for
```

Método base On Web Close Process

Método base On Web Close Process
Este comando no requiere parámetros

---

*El **Método base On Web Close Process** es llamado por el servidor web de 4D cada vez que una sesión web se va a cerrar. Una sesión puede ser cerrarse en los siguientes casos:*

- *cuando se alcanza el número máximo de sesiones simultáneas (100 por defecto, modificable utilizando el comando **WEB SET OPTION**), y 4D necesita crear nuevas (4D automáticamente destruye el proceso de la sesión inactiva más antigua),*
- *cuando se alcanza el periodo máximo de inactividad del proceso de la sesión (480 minutos por defecto, modificable vía el comando **WEB SET OPTION**),*
- *cuando se llama al comando **WEB CLOSE SESSION**.*

*Cuando se llama a este método base, el contexto de la sesión (variables y selecciones generadas por el usuario) es aún valido. Esto significa que puede guardar los datos relativos a la sesión para poder usarlos posteriormente, más específicamente utilizando **Método base On Web Connection**.*
***Nota:** en el contexto de una sesión 4D Mobile (que puede generar varios procesos), el **Método base On Web Close Process** se llama para cada proceso web cerrado, lo que permite guardar todo tipo de datos (variables, selección, etc.) generados por el proceso de sesión 4D Mobile.*

*Un ejemplo de uso del **Método base On Web Close Process** se presenta en la sección **Gestión de las sesiones web**.*

Método base On Web Connection

$1, $2, $3, $4, $5, $6 -> Método base On Web Connection

Parámetro	Tipo		Descripción
$1	Texto	⬅	URL
$2	Texto	⬅	Encabezado HTTP + Cuerpo HTTP
$3	Texto	⬅	Dirección IP del navegador
$4	Texto	⬅	Dirección IP del servidor
$5	Texto	⬅	Nombre de usuario
$6	Texto	⬅	Contraseña

---

*El **Método base On Web Connection** puede llamarse en los siguientes casos:*

- *el servidor web recibe una petición que comienza por el URL 4DCGI.*
- *el servidor web recibe una petición inválida.*

*Para mayor información, consulte el párrafo "**Llamadas al Método de base On Web Connection**" abajo.*
*La petición debe haber sido aceptada previamente por el **Método base On Web Authentication** (si existe) y el servidor web debe lanzarse.*
*El **Método base On Web Connection** recibe seis parámetros de tipo texto, pasados por 4D ($1, $2, $3, $4, $5 y $6). Los contenidos de estos parámetros son los siguientes:*

Parámetros	Tipo	Descripción
$1	Texto	URL
$2	Texto	Encabezado HTTP + cuerpo HTTP (hasta 32 kb de límite)
$3	Texto	Dirección IP del navegador
$4	Texto	Dirección IP llamada del servidor
$5	Texto	Nombre de usuario
$6	Texto	Contraseña

Debe declarar estos seis parámetros de esta manera:

```
` Método de base On Web Connection

C_TEXT($1;$2;$3;$4;$5;$6)

` Código para el método
```

- ***Datos extra del URL***

  *El primer parámetro ($1) es el **URL** introducido por el usuario en el área de ubicación de su navegador web, menos la dirección local.*
  *Tomemos el ejemplo de una conexión de Intranet. Supongamos que la dirección **IP** de su equipo servidor web 4D es 123.4.567.89. La tabla siguiente muestra los valores de $1 dependiendo del URL introducido en el navegador web:*

URL introducido en el navegador	Valor del parámetro $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clientes	/Clientes
http://123.4.567.89/Clientes	/Clientes
http://123.4.567.89/Clientes/Añadir	/Clientes/Añadir
123.4.567.89/Hacer_esto/Si_OK/Hacer_eso	/Hacer_esto/Si_OK/Hacer_eso

  Note que usted es libre de utilizar este parámetro a su conveniencia. 4D simplemente ignora los valores pasados más allá de la parte local del URL. Por ejemplo, puede establecer una convención donde el valor "/Clientes/Añadir" signifique "ir directamente a añadir un nuevo registro en la tabla [Clientes]." Suministrando a los usuarios web de su base una lista de posibles valores y/o marcadores por defecto, puede ofrecer atajos a las diferentes partes de su aplicación. De esta forma, los usuarios web pueden acceder rápidamente a los recursos de su sitio web sin tener que navegar cada vez que se conecten a su base.

***Advertencia:** para evitar que un usuario acceda directamente a una base con un marcador creado durante una sesión anterior, 4D intercepta todo URL que corresponda a uno de los URLs estándar de 4D.*

- ***Encabezado y cuerpo de la petición HTTP***
  *El segundo parámetro ($2) es el encabezado y el cuerpo de la petición HTTP enviada por el navegador web. Note que esta información se pasa a su **Método base On Web Connection** tal como está. El contenido varía en función del tipo de navegador web que esté intentando la conexión*
  Con Safari corriendo en Mac OS, puede recibir un encabezado similar a este:

```
GET /favicon.ico HTTP/1.1
Referer: http://123.45.67.89/4dcgi/test
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; fr-fr) AppleWebKit/523.10.3 (KHTML, like Gecko) Version/3.0.4
Safari/523.10
Cache-Control: max-age=0
Accept: */*
Accept-Language: fr-fr
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: 123.45.67.89
```

Con Microsoft Internet Edge en Windows, puede recibir un encabezado similar a este:

```
GET / HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg, application/x-ms-xbap,
application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: fr-FR
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729;
.NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C)
Accept-Encoding: gzip, deflate
Host: 123.45.67.89
Connection: Keep-Alive
```

Si su aplicación manipula esta información, es su decisión analizar el encabezado y el cuerpo.
***Nota:** por razones de rendimiento, el tamaño de estos datos no puede ser mayor a 32 KB. Si el tamaño es mayor, los datos son truncados por el servidor 4D HTTP.*

- ***Dirección IP del cliente web***
  El tercer parámetro $3 recibe la dirección IP del equipo navegador. Esta información puede permitirle distinguir entre las conexiones de Intranet e Internet.
  *Nota**:** 4D devuelve direcciones IPv4 en un formato híbrido IPv6 escrito con un prefijo de 96 bits, por ejemplo ::ffff:192.168.2.34 para la dirección IPv4 192.168.2.34. Para mayor información, consulte la sección* ***Soporte de IP v6****.*

- ***Dirección IP del servidor***
  El cuarto parámetro $4 recibe la dirección IP solicitada del servidor web 4D. 4D autoriza el multi-homing, el cual le permite explotar equipos que tengan más de una dirección IP. Para mayor información, consulte la sección ***Parámetros del servidor web****.*

- ***Nombre de usuario y contraseña***
  Los parámetros $5 y $6 reciben el nombre de usuario y la contraseña introducidos por el usuario en la caja de diálogo estándar de identificación mostrada por el navegador. Esta caja de diálogo aparece para cada conexión, si la opción ***Utilizar contraseñas*** *ha sido seleccionada en las Preferencias (ver sección* ***Seguridad de las conexiones****).*

***Nota:** si el nombre de usuario enviado por el navegador existe en 4D, el parámetro $6 (la contraseña del usuario) no se devuelve por razones de seguridad.*

Llamadas al Método de base On Web Connection

El ***Método base On Web Connection*** *puede utilizarse como punto de entrada para el servidor web 4D, bien sea utilizando el URL especial 4DCGI, o utilizando los URLs de comando personalizados.*

***Advertencia:** la llamada de un comando 4D que muestra un elemento de interfaz (**DIALOG**, **ALERT**...) termina el procesamiento del método.*

El ***Método base On Web Connection*** *se llama en los siguientes casos:*

- *Cuando 4D recibe el URL /4DCGI. El método base se llama con el URL /4DCGI/<action> en $1.*
- *Cuando una página web llamada con un URL de tipo <ruta>/<archivo> no se encuentra. El método de base se llama con el URL (*).*
- *Cuando una página web se llama con un URL del tipo <file>/ y ninguna página ha sido definida por defecto. El método de base se llama con el URL (*).*

(*) En estos casos particulares, el URL recibido en $1 NO comienza por el carácter "/".

Objetos (Formularios)

Áreas 4D Write Pro

*La mayoría de los comandos del tema "**Objetos (formularios)**" soportan áreas 4D Write Pro. Para obtener más información al respecto, consulte la sección **Utilizar los comandos del tema Objeto (Formularios)** en el manual 4D Write Pro Reference.*

- *Propiedades de los objetos*
- *GET STYLE SHEET INFO*
- *LIST OF STYLE SHEETS*
- *OBJECT DUPLICATE*
- *OBJECT Get action*
- *OBJECT Get auto spellcheck*
- *OBJECT GET BEST SIZE*
- *OBJECT Get border style*
- *OBJECT Get context menu*
- *OBJECT GET COORDINATES*
- *OBJECT Get corner radius*
- *OBJECT Get data source*
- *OBJECT GET DRAG AND DROP OPTIONS*
- *OBJECT Get enabled*
- *OBJECT Get enterable*
- *OBJECT GET EVENTS*
- *OBJECT Get filter*
- *OBJECT Get focus rectangle invisible*
- *OBJECT Get font*
- *OBJECT Get font size*
- *OBJECT Get font style*
- *OBJECT Get format*
- *OBJECT Get help tip*
- *OBJECT Get horizontal alignment*
- *OBJECT Get indicator type*
- *OBJECT Get keyboard layout*
- *OBJECT Get list name*
- *OBJECT Get list reference*
- *OBJECT GET MAXIMUM VALUE*
- *OBJECT GET MINIMUM VALUE*
- *OBJECT Get multiline*
- *OBJECT Get name*
- *OBJECT Get placeholder*
- *OBJECT Get pointer*
- *OBJECT GET PRINT VARIABLE FRAME*
- *OBJECT GET RESIZING OPTIONS*
- *OBJECT GET RGB COLORS*
- *OBJECT GET SCROLL POSITION*
- *OBJECT GET SCROLLBAR*
- *OBJECT GET SHORTCUT*
- *OBJECT Get style sheet*
- *OBJECT GET SUBFORM*
- *OBJECT GET SUBFORM CONTAINER SIZE*
- *OBJECT Get text orientation*
- *OBJECT Get three states checkbox*
- *OBJECT Get title*
- *OBJECT Get type*
- *OBJECT Get vertical alignment*
- *OBJECT Get visible*
- *OBJECT Is styled text*
- *OBJECT MOVE*
- *OBJECT SET ACTION*
- *OBJECT SET AUTO SPELLCHECK*
- *OBJECT SET BORDER STYLE*
- *OBJECT SET COLOR*
- *OBJECT SET CONTEXT MENU*
- *OBJECT SET COORDINATES*
- *OBJECT SET CORNER RADIUS*
- *OBJECT SET DATA SOURCE*
- *OBJECT SET DRAG AND DROP OPTIONS*
- *OBJECT SET ENABLED*
- *OBJECT SET ENTERABLE*
- *OBJECT SET EVENTS*
- *OBJECT SET FILTER*

- *OBJECT SET FOCUS RECTANGLE INVISIBLE*
- *OBJECT SET FONT*
- *OBJECT SET FONT SIZE*
- *OBJECT SET FONT STYLE*
- *OBJECT SET FORMAT*
- *OBJECT SET HELP TIP*
- *OBJECT SET HORIZONTAL ALIGNMENT*
- *OBJECT SET INDICATOR TYPE*
- *OBJECT SET KEYBOARD LAYOUT*
- *OBJECT SET LIST BY NAME*
- *OBJECT SET LIST BY REFERENCE*
- *OBJECT SET MAXIMUM VALUE*
- *OBJECT SET MINIMUM VALUE*
- *OBJECT SET MULTILINE*
- *OBJECT SET PLACEHOLDER*
- *OBJECT SET PRINT VARIABLE FRAME*
- *OBJECT SET RESIZING OPTIONS*
- *OBJECT SET RGB COLORS*
- *OBJECT SET SCROLL POSITION*
- *OBJECT SET SCROLLBAR*
- *OBJECT SET SHORTCUT*
- *OBJECT SET STYLE SHEET*
- *OBJECT SET SUBFORM*
- *OBJECT SET TEXT ORIENTATION*
- *OBJECT SET THREE STATES CHECKBOX*
- *OBJECT SET TITLE*
- *OBJECT SET VERTICAL ALIGNMENT*
- *OBJECT SET VISIBLE*
- *_o_DISABLE BUTTON*
- *_o_ENABLE BUTTON*
- *_o_OBJECT Get action*

Propiedades de los objetos

Los comandos de propiedades de los objetos actúan en las propiedades de los objetos presentes en los formularios. Estos comandos le permiten cambiar la apariencia y el comportamiento de los objetos mientras utiliza los formularios en el entorno de aplicación.

***Importante**: el alcance de estos comandos es el formulario que está siendo utilizado; los cambios desaparecen al salir del formulario.*

Acceso a los objetos por sus nombres de objeto o por sus nombres de fuentes de datos

Los comandos de propiedades de los objetos comparten la misma sintaxis genérica:

***NOMBRE DEL COMANDO**({*;} objeto { ; parámetros adicionales específicos para cada comando)*

*Si especifica el parámetro opcional *, usted indica un nombre de objeto (una cadena) en objeto.*

***Nota**: es posible utilizar el carácter @ en ese nombre si quiere direccionar varios objetos de un formulario en una sola llamada. La siguiente tabla muestra ejemplos de nombres de objetos que usted puede especificar para este comando.*

Nombres de objetos	Objetos afectados por la llamada
areaGrupo	Únicamente el objeto areaGrupo.
area@	Los objetos cuyo nombre comienza por "area".
@areaGrupo	Los objetos cuyo nombre termina en "areaGrupo".
@Grupo@	Los objetos cuyo nombre contiene "Grupo".
area@Btn	Los objetos cuyo nombre comienza por "area" y termina en "Btn".
@	Todos los objetos presentes en el formulario.

Los nombres de objetos de formulario pueden contener hasta 255 bytes, permitiéndole definir y aplicar las reglas de nomenclatura personalizada, como "xxxx_Button" o "xxx_Mac".

***Nota**: es posible configurar la forma en que el carácter @ es interpretado cuando éste se incluye en una cadena de caracteres. Esta opción afecta el funcionamiento de los comandos del tema "Objetos (Formularios)". Para más información, consulte el manual de Diseño 4D.*

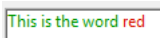
*Si omite el parámetro opcional *, indica un campo o variable en objeto. En este caso, usted especifica una referencia de campo o de variable (campo y variable objeto únicamente) en lugar de una cadena.*

Interacción de comandos genéricos con áreas multiestilos

*A partir de 4D v14, un nuevo modo de interacción se ha definido entre los comandos genéricos tales como **OBJECT SET RGB COLORS** o **OBJECT SET FONT STYLE** y las áreas de texto multi-estilo.*

En las versiones anteriores de 4D, la ejecución de uno de estos comandos modificaba el contenido de las etiquetas de estilo personalizadas, insertadas en el área. A partir de ahora, sólo las propiedades por defecto se ven afectadas por estos comandos (así como las propiedades guardadas vía las etiquetas por defecto). Las etiquetas de estilo personalizadas se quedan como están.

Por ejemplo, dada un área multi-estilo, donde se guardan las etiquetas por defecto:



El texto plano del área es el siguiente:

```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#009900">This is the word <span
style="color:#D81E05">rojo</span></span>
```

Si ejecuta el siguiente código:

```
OBJECT SET COLOR(*;"myArea";-(Blue+(256*Yellow)))
```

Con 4D v14, el color rojo se mantiene:

4D v14

```
<span style="text-align:left;font-family:'Segoe
UI';font-size:9pt;color:#0000FF">This is the
word <span style="color:#D81E05">red</span>
</span>
```

versiones anteriores

```
<span style="font-family:'Segoe UI';font-size:9pt;text-align:left;font-
weight:normal;font-style:normal;text-decoration:none;color:#0000FF;">
<span style="background-color:#FFFFFF">This is the red word</span>
</span>
```

Los siguientes comandos genéricos están relacionados:

OBJECT SET RGB COLORS
OBJECT SET COLOR
OBJECT SET FONT
OBJECT SET FONT STYLE
OBJECT SET FONT SIZE

*En el contexto de las áreas de texto multiestilos, los comandos genéricos deben utilizarse sólo para definir estilos por defecto. Para administrar estilos durante la ejecución de las base, recomendamos utilizar los comandos del tema "**Texto multiestilo**".*

GET STYLE SHEET INFO

GET STYLE SHEET INFO (nomHojaEstilo ; fuente ; tam ; estilos)

Parámetro	Tipo		Descripción
nomHojaEstilo	Texto	➡	Nombre de la hoja de estilo
fuente	Texto	⬅	Tipo de fuente
tam	Entero largo	⬅	Tamaño de fuente
estilos	Entero largo	⬅	Valor del estilo

Descripción

*El comando **GET STYLE SHEET INFO** devuelve la configuración actual de la hoja de estilo nomHojaEstilo .*

*Pase en nomHojaEstilo, el nombre de la hoja de estilo definida en modo Diseño. Para designar una hoja de estilo Automática, utilice una de las siguientes constantes, ubicadas en el tema "**Estilos de fuente**":*

Constante	Tipo	Valor	Comentario
Automatic style sheet	Cadena	__automatic__	Se utiliza de forma predeterminada para todos los objetos
Automatic style sheet_additional	Cadena	__automatic_additional_text__	Soportado por los textos estáticos, campos y variables solamente. Se utiliza para texto adicional en las cajas de diálogo.
Automatic style sheet_main	Cadena	__automatic_main_text__	Soportado por los textos estáticos, campos y variables solamente. Se utiliza para texto principal en las cajas de diálogo.

El comando devuelve en fuente, el nombre de la fuente de caracteres asociada a la hoja de estilo para la plataforma actual.

El comando devuelve en tam, el tamaño en puntos de la fuente asociada a la hoja de estilo para la plataforma actual.

*El comando devuelve en estilos, un valor que corresponde al estilo(s) asociado(s) a la hoja de estilo para la plataforma actual. Puede comparar el valor recibido con las siguientes constantes, que se encuentran en el tema "**Estilos de fuente**":*

Constante	Tipo	Valor
Bold	Entero largo	1
Bold and Italic	Entero largo	3
Bold and Underline	Entero largo	5
Italic	Entero largo	2
Italic and Underline	Entero largo	6
Plain	Entero largo	0
Underline	Entero largo	4

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario (por ejemplo, si nomHojaEstilo no existe), toma el valor 0.

Ejemplo

Si quiere conocer la configuración actual de la hoja de estilo "Automatic"

```
C_LONGINT($size;$style)
C_TEXT($font)
GET STYLE SHEET INFO(Automatic style sheet;$font;$size;$style)
```

LIST OF STYLE SHEETS

LIST OF STYLE SHEETS (arrHojasEstilo)

Parámetro	*Tipo*		*Descripción*
arrHojasEstilo	*Array texto*	⬅	*Nombres de las hojas de estilo definidas en la aplicación*

Descripción

*El comando **LIST OF STYLE SHEETS** devuelve la lista de hojas de estilo de la aplicación en el array arrHojasEstilo.*

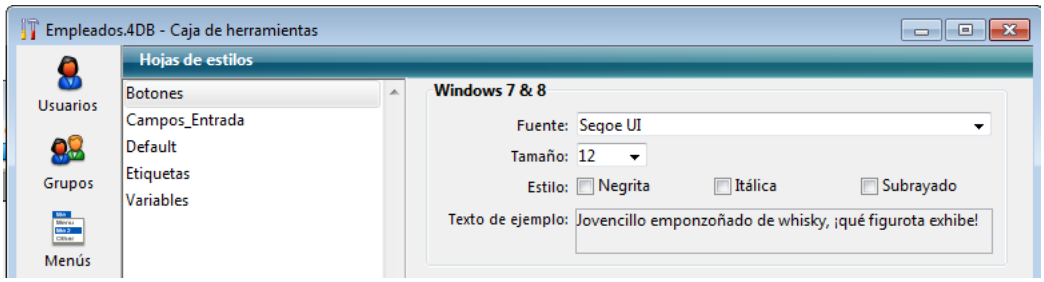
Si no hubiera sido previamente definido, el array arrHojasEstilo es creado por el comando. Se dimensiona automáticamente de acuerdo con el número de hojas de estilo definidas.

Después de ejecutar el comando, cada elemento del array contiene el nombre de una hoja de estilos. Estos nombres se ordenan alfabéticamente, como en el editor de hojas de estilo. El primer elemento del array contiene siempre "__automatic__", que representa la hoja de estilo "Automática".

***Nota**: por razones de compatibilidad, las hojas de estilo automáticas "__automatic_main_text__" y "__automatic_additional_text__" no son devueltas por este comando. Sin embargo, todavía están disponibles en los formularios.*

Ejemplo

En su aplicación, se definen las siguientes hojas de estilo:



Si ejecuta el siguiente código:

```
LIST OF STYLE SHEETS($arrStyles)
 // $arrStyles{1} contiene "__automatic__"
 // $arrStyles{2} contiene "Buttons"
 // $arrStyles{3} contiene "default"
 // $arrStyles{4} contiene "Input_fields"
 // $arrStyles{5} contiene "Labels"
 // $arrStyles{6} contiene "Variables"
```

OBJECT DUPLICATE

OBJECT DUPLICATE ({* ;} objeto {; nuevoNom {; nuevaVAr {; relacionadoA {; movH {; moveV {; redimH {; redimV}}}}}}} {; *})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o variable o campo (si se omite *)
nuevoNom	Texto	➡	Nombre del nuevo objeto
nuevaVAr	Puntero	➡	Puntero a la variable del nuevo objeto
relacionadoA	Texto	➡	Nombre del objeto editable (o del botón de radio) anterior
movH	Entero largo	➡	Desplazamiento horizontal del nuevo objeto (>0 = a la derecha, <0 =a la izquierda)
moveV	Entero largo	➡	Desplazamiento vertical del nuevo objeto (>0 = hacia abajo, <0= hacia arriba)
redimH	Entero largo	➡	Valor de redimensionamiento horizontal del nuevo objeto
redimV	Entero largo	➡	Valor de redimensionamiento vertical del nuevo objeto
*	Operador	➡	Si se especifica = coordenadas absolutas Si se omite = coordenadas relativas

Descripción

El comando OBJECT DUPLICATE permite crear una copia del objeto designado por el parámetro objeto en el contexto del formulario que está siendo ejecutado (modo Aplicación). El formulario de origen, generado en modo Diseño, no se modifica.

Por defecto, todas las opciones definidas en la lista de propiedades para el objeto fuente se aplican a la copia (tamaño, opciones de redimensionamiento, color, etc.), incluyendo el método de objeto asociado. Sin embargo, se deben tener en cuenta las siguientes excepciones:

- Botón por defecto: sólo puede haber un botón por defecto en un formulario. Al duplicar un botón con la propiedad "Botón por defecto", esta propiedad se asigna a la copia y se suprime del objeto de origen.
- Equivalentes de teclado: el atajo de teclado asociado a un objeto fuente no se duplica. Esta propiedad se deja en blanco en la copia.
- Nombres de objetos: no puede haber varios objetos con el mismo nombre en un formulario. Si no pasa el parámetro nuevoNom, el nombre del objeto fuente automáticamente se incrementa en el nuevo objeto (ver a continuación).

Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, pase una referencia del campo o variable (campos o variables objeto únicamente) en lugar de una cadena.
Si pasa una referencia de campo o variable y si el formulario contiene varios objetos que utilizan la misma referencia, la primera ocurrencia encontrada se utilizará. En este caso, para evitar cualquier ambigüedad, se recomienda trabajar con nombres de objetos únicos.

Pase en el parámetro nuevoNom el nombre asignado a la copia del objeto. Este nombre debe ser conforme con la reglas de nombre de objetos y ser único en el formulario. Si no es válido o ya está siendo utilizado por otro objeto, el comando no hace nada y la variable OK devuelve 0.
Si omite este parámetro o pasa una cadena vacía, el nuevo nombre se genera automáticamente al incrementar el nombre del objeto fuente (si este nombre no se utiliza). Por ejemplo:

Nombre de origen	Nombre de la copia
Botón	Botón1
Botón20	Botón21
Botón21	Botón23 si Botón22 ya existe

Pase en nuevaVar un puntero a la variable a asociar al nuevo objeto. En principio, debe estar dirigido a una variable del mismo tipo que el del objeto fuente pero en algunos casos es posible cambiar el tipo. El comando ofrece funciones automáticas para facilitar la escritura de código genérico:

- Por lo general, se puede cambiar el tipo a todas las variables "editables"; por ejemplo, un objeto que muestra una Fecha o un Entero largo puede duplicarse y utilizarse con una variable de tipo Texto. Las propiedades compatibles se conservan. El comando permite igualmente cambiar el tipo entre objetos Texto y objetos Imagen. Note que un objeto texto duplicado y asociado a una variable o campo Booleano aparecerá automáticamente como casilla de selección.
- Por lo general es posible transformar dinámicamente un variable en campo y viceversa.
  Por otra parte, los objetos gráficos (botones, casillas de selección, etc.) no pueden transformarse en otros tipos de controles.

Si el tipo de la variable no es compatible con el objeto, el comando no hace nada y la variable OK toma el valor 0. Si omite este parámetro, la variable es creada dinámicamente por 4D (ver el párrafo "Variables dinámicas" en la sección). Si duplica un objeto estático (líneas, rectángulo, imagen estática, etc.), este parámetro se ignora. Pase un puntero Nil (->[]) si quiere poder utilizar los otros parámetros.

Utilice el parámetro relacionadoA en dos casos:

- actualizar el orden de entrada: en este caso, en relacionadoA, pase el nombre del objeto editable ubicado justo antes del objeto duplicado. Si quiere que el nuevo objeto se convierta en el primer objeto en el orden de entrada de la página, pase la constante Object First in entry order (ver el comando **OBJECT Get pointer**).
- asociación con un grupo de botones radio: los botones radio funcionan de manera coordinada cuando están agrupados. Si el objeto duplicado es un botón de radio, en relacionadoA pase el nombre de un botón radio del grupo al cual asociar el nuevo objeto.

Si omite este parámetro o pasa una cadena vacía, el nuevo objeto se convierte en el último objeto editable de la página del formulario.

*El nuevo objeto puede moverse y redimensionarse por medio de los parámetros moveH, moveV, redimH y redimV. Como para el comando **OBJECT MOVE**, el sentido de desplazamiento o redimensionamiento es definido por el signo de los valores pasados en los parámetros moveH y moveV:*

- *Si el valor es positivo, el desplazamiento o redimensionamiento se efectúa respectivamente hacia la derecha o hacia abajo.*
- *Si el valor es negativo, el movimiento o redimensionamiento se efectúa respectivamente hacia la izquierda o hacia arriba.*

*Por defecto, los valores de moveH, moveV, redimH y redimV modifican las coordenadas del objeto en relación con su posición anterior. Si quiere que estos parámetros especifiquen coordenadas absolutas, pase el parámetro opcional *.*
Si omite estos parámetros, el nuevo objeto se superpone al objeto de origen.

Este comando debe utilizarse en el contexto de la visualización de un formulario. Por lo general se llama en el evento On Load del formulario o luego de una acción usuario (evento On Clicked).

***Nota:** si el evento On Load está asociado al objeto de origen, se genera para el objeto duplicado al ejecutar el comando. Esto permite, por ejemplo, inicializar el valor del objeto.*

*Por razones técnicas y lógicas, **OBJECT DUPLICATE** no puede llamarse dentro de ciertos eventos de formulario, en particular:*

- *Evento On Load generado en un método de objeto*
- *Evento On Unload.*

Cuando el comando se llama en un contexto no soportado, el objeto no se duplica y la variable OK toma el valor 0. Si se llama en un contexto de impresión, el error -10601 se genera también.

Si el comando se ejecutó correctamente, la variable OK toma el valor 1. De lo contrario toma el valor 0.

Ejemplo 1

Creación de un nuevo botón llamado "BotonCancel" sobre el objeto existente "BotónOK" y asociación con la variable vCancel:

```
OBJECT DUPLICATE(*;"BotonOK";"BotonCancel";vCancel)
```

Ejemplo 2

Creación de un nuevo botón radio "bRadio6" basado en el botón radio existente "bRadio5". Este botón se asociará a la variable <>r6, integrada con el grupo del botón "bRadio5" y ubicado 20 píxeles arriba:

```
OBJECT DUPLICATE(*;"bRadio5";"bRadio6";<>r6;"bRadio5";0;20)
```

OBJECT Get action

OBJECT Get action ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Campo o variable (si se omite *)
Resultado	Texto	➲	Nombre de la acción estándar asociada y (si la hay) cadena de parámetros

Descripción

El comando ***OBJECT Get action*** *devuelve el nombre y (si es el caso) el parámetro de la acción estándar asociada con el objeto designado por los parámetros objeto y *.*

*Pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (solo objeto campo o variable).*

Puede establecer una acción estándar para un objeto en el Editor de formularios utilizando la lista de propiedades o utilizando el comando ***OBJECT SET ACTION****.* ***OBJECT Get action*** *devuelve una cadena que contiene el nombre de la acción estándar asociada al objeto (así como su parámetro, si existe).*

Para obtener una lista completa de acciones estándar, consulte la sección ***Acciones estándar*** *en el manual de Diseño.*

Ejemplo

Usted desea asociar la acción "Cancelar" con todos los objetos en el formulario que aún no tienen ninguna acción asociada:

```
ARRAY TEXT($arrObjects;0)

FORM GET OBJECTS($arrObjects)
For($i;1;Size of array($arrObjects))
   If(OBJECT Get action(*;$arrObjects{$i})=ak none)
      OBJECT SET ACTION(*;$arrObjects{$i};ak cancel)
   End if
End for
```

OBJECT Get auto spellcheck

OBJECT Get auto spellcheck ({* ;} objeto) -> Resultado

Parámetro	*Tipo*		*Descripción*
*	*Operador*	➡	*Si se especifica, objeto es un nombre de objeto (cadena)* *Si se omite, objeto es una variable o campo*
objeto	*Objeto de formulario*	➡	*Nombre del objeto (si se especifica *) o* *Variable o campo(si se omite *)*
Resultado	*Booleano*	⮌	*True = corrección automática,* *False = no corrección automática*

Descripción

*El comando **OBJECT Get auto spellcheck** devuelve el estado de la opción Corrección ortográfica automática del o de los objeto(s) designado(s) por los parámetros objeto y * para el proceso actual .*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que objeto es una variable o un campo. En este caso, pase una referencia en lugar de un nombre.*

*El comando devuelve **True** cuando la corrección ortográfica automática está activada para el objeto y **False** si no.*

OBJECT GET BEST SIZE

OBJECT GET BEST SIZE ({* ;} objeto ; largOpt ; altOpt {; anchoMax})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre del objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
largOpt	Entero largo	⬅	Largo óptimo del objeto
altOpt	Entero largo	⬅	Alto óptimo del objeto
anchoMax	Entero largo	➡	Largo máximo del objeto

Descripción

El comando ***OBJECT GET BEST SIZE*** *devuelve en los parámetros largOpt y altOpt, el largo y alto "óptimo" del objeto de formulario designado por los parámetros * y objeto. Estos valores se expresan en píxeles. Este comando es particularmente útil para la visualización o impresión de informes complejos, asociados al comando* ***OBJECT MOVE****.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena de caracteres). Si no pasa el parámetro *, indica que objeto es un campo o una variable. En este caso, usted no pasa una cadena sino una referencia de un campo o de una variable (de tipo objeto únicamente).*

Los valores óptimos devueltos indican el tamaño mínimo del objeto de manera que su contenido actual sea incluido completamente en los límites. Por o general estos valores sólo son significativos para objetos que contengan texto. Este cálculo tiene en cuenta la fuente, su tamaño, estilo y contenido del objeto. También tiene en cuenta la incorporación de guiones y de retornos de carro.

Note que en el caso de los botones 3D, el funciona igual si el botón contiene únicamente un ícono.

Si el objeto especificado está vacío, el largOpt devuelto es 0.

El tamaño devuelto no tiene en cuenta marcos de gráficos aplicados alrededor del objeto, ni barras de desplazamiento. Para obtener el tamaño real de un objeto en pantalla, es necesario añadir el largo de estos elementos.

El parámetro opcional largoMax le permite atribuir un largo máximo al objeto. Si el largo óptimo del objeto es superior a este valor, ***OBJECT GET BEST SIZE*** *devuelve largoMax en el parámetro largoOpt y aumenta el alto óptimo en consecuencia.*

Los siguientes objetos son manejados por este comando:

- *Áreas de texto estáticas*
- *Textos insertados en forma de referencias*
- *Campos y variables de los siguientes tipos: Alfa, Texto, Real, Entero, Entero largo, Fecha, Hora, Booleano (casillas de selección y botones de radio)*
- *Botones*
- *Columnas de list box en contexto de visualización (sólo las líneas visibles se tienen en cuenta).*

Para todos los otros tipos de objetos de formulario (áreas de grupos, pestañas, rectángulos, líneas rectas, círculos/óvalos, áreas externas, etc.), el comando ***OBJECT GET BEST SIZE*** *devuelve el tamaño del objeto actual (definido en el editor de formularios y eventualmente utilizando el comando* ***OBJECT MOVE****).*

Ejemplo

Consulte el ejemplo en el comando ***SET PRINT MARKER****.*

OBJECT Get border style

OBJECT Get border style ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
Resultado	Entero largo	⮌	Estilo de línea de borde

Descripción

El comando ***OBJECT Get border style*** *devuelve el estilo de línea de borde del objeto o de los objetos designado(s) por los parámetros objeto y *.*

Puede definir el estilo de línea del borde para un objeto en modo de diseño utilizando la lista de propiedades, o utilizando el comando ***OBJECT SET BORDER STYLE****.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*El comando devuelve un valor que corresponde al estilo de la línea fronteriza. Puede comparar el valor recibido con las siguientes constantes, que se encuentran en el tema "****Propiedades de los objetos****":*

Constante	Tipo	Valor	Comentario
Border Dotted	Entero largo	2	Los objetos aparecen enmarcados con una línea punteada de 1-pt.
Border Double	Entero largo	5	Los objetos aparecen enmarcados con una línea doble, es decir, dos líneas continuas de 1-pt. separadas por un píxel
Border None	Entero largo	0	Los objetos aparecen sin borde
Border Plain	Entero largo	1	Los objetos aparecen enmarcado con una línea de borde continua de 1-pt.
Border Raised	Entero largo	3	Los objetos aparecen con un efecto 3D (relieve)
Border Sunken	Entero largo	4	Los objetos aparecen enmarcados con un efecto 3D hundido (relieve inverso)
Border System	Entero largo	6	La línea del borde se dibuja en función de las especificaciones gráficas del sistema

OBJECT Get context menu

OBJECT Get context menu ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
Resultado	Booleano	⮌	True = menu contextual activo, False = menu contextual inactivo

Descripción

El comando ***OBJECT Get context menu*** *devuelve el estado actual de la opción "Menú contextual" del objeto o de los objetos designado(s) por los parámetros objeto y * .*

Puede configurar la opción "Menú contextual" en modo Diseño utilizando la lista de propiedades o utilizando el comando ***OBJECT SET CONTEXT MENU****.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

El comando devuelve ***True*** *si el menú contextual está activo para el objeto y* ***False*** *en caso contrario.*

OBJECT GET COORDINATES

OBJECT GET COORDINATES ({* ;} objeto ; izquierdo ; superior ; derecho ; inferior)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es el nombre del objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
izquierdo	Entero largo	⬅	Coordenada izquierada del objeto
superior	Entero largo	⬅	Coordenada superior del objeto
derecho	Entero largo	⬅	Coordenada derecha del objeto
inferior	Entero largo	⬅	Coordenada inferior del objeto

Descripción

*El comando **OBJECT GET COORDINATES**devuelve las coordenadas izquierda, superior, derecha e inferior (en puntos) en las variables o campos de los objetos del formulario actual definido por los parámetros * y objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena de caracteres). Si no pasa el parámetro opcional *, indica que objeto es un campo o una variable. En este caso, no se pasa una cadena sino una referencia de un campo o de una variable (campo o variable de tipo objeto únicamente).*

Si pasa un nombre de objeto en el parámetro objeto y utiliza el carácter arroba ("@") para seleccionar más de un objeto, las coordenadas devueltas serán las del rectángulo formado por todos los objetos concernientes.

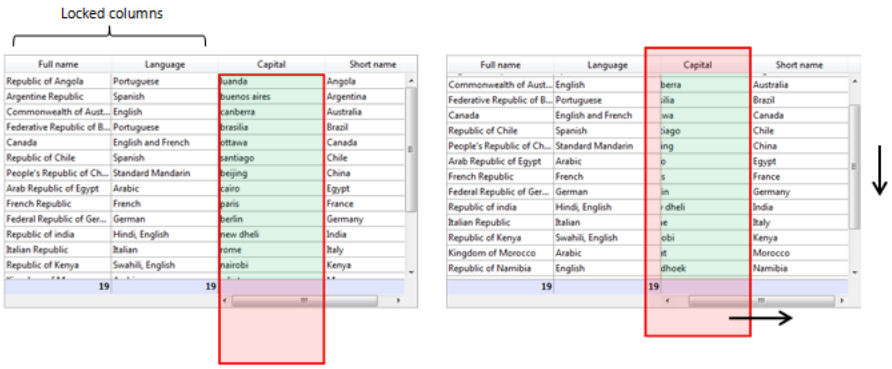
***Nota:** desde la versión 6.5, es posible definir el modo de interpretación del carácter arroba ("@"), cuando se incluye en una cadena de caracteres. Esta opción tiene un impacto en los comandos "Propiedades de los objetos". Por favor consulte el Manual de Diseño.*

Si el objeto no existe o si no se llama el comando en el contexto de un formulario, las coordenadas devueltas son (0;0;0;0).

*En el contexto de los list box, el comando **OBJECT GET COORDINATES** puede devolver las coordenadas de partes específicas del listbox, es decir, columnas, encabezados, pies de página, y no sólo los del objeto listbox padre. En las versiones anteriores a v14 R5, este comando siempre devuelve las coordenadas del listbox padre, independientemente del área pasada como parámetro. A partir de ahora, cuando el objeto referenciado es un encabezado, una columna o un pie de listbox, las coordenadas devueltas son los del sub-objeto listbox designado. Puede utilizar esta nueva funcionalidad, por ejemplo, para mostrar un pequeño icono en la celda de encabezado de un listbox cuando se pasa sobre él, indicándole al usuario que puede hacer clic para mostrar un menú contextual.*
Para mantener la coherencia, el marco de referencia utilizado es el mismo cuando el objeto es un sub-objeto list box o un objeto list box: el origen es la esquina superior izquierda del formulario que contiene el objeto. Para los sub-objetos de listbox, las coordenadas devueltas son teóricas; tienen en cuenta el estado de desplazamiento del list box antes de que ocurra un clipping (es decir, el corte de acuerdo con las coordenadas del list box padre). Como resultado, el sub-objeto puede que no sea visible (o sólo parcialmente) en sus coordenadas, y estas coordenadas pueden estar fuera de los límites del formulario (o incluso ser negativas). Para averiguar si el sub-objeto es visible (y que parte es visible) es necesario comparar las coordenadas devueltas con las coordenadas del listbox, mientras tiene en cuenta las siguientes reglas:

- *Todos los sub-objetos se recortan según las coordenadas de su listbox padre (devueltas por **OBJECT GET COORDINATES** en el list box).*
- *Los sub-objetos encabezado y pie se muestran sobre el contenido de la columna: cuando las coordenadas de una columna cruzan las coordenadas de las líneas de encabezado o pie de página, luego la columna no se muestra en esta intersección.*
- *Los elementos de las columnas bloqueadas se muestran arriba de los elementos de las columnas desplazables: cuando las coordenadas de un elemento en una columna desplazable cruzan las coordenadas de un elemento en una columna bloqueada, no se muestra en esta intersección.*

Por ejemplo, considere el siguiente gráfico, donde las coordenadas de la columna Capital están simbolizadas por un rectángulo rojo:



Como se puede ver en la primera imagen, la columna es más grande que el listbox, por lo que sus coordenadas van más allá del límite inferior del listbox, incluyendo el pie de página. En la segunda imagen, el listbox se ha desplazado, por lo que la columna también se ha movido "bajo" la columna Language y el área de encabezado. En cualquier caso, con el fin de calcular la parte visible real (área verde), es necesario sustraer las áreas rojas.

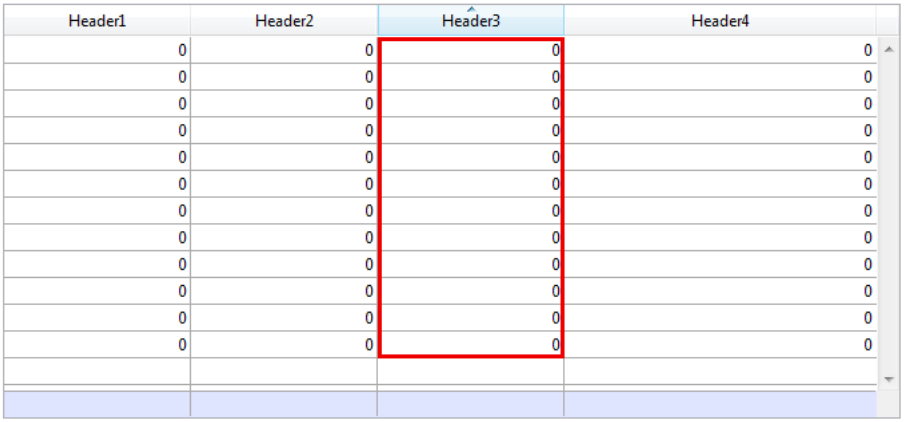
Ejemplo 1

Asumamos que quiere obtener las coordenadas de un rectángulo formado por todos los objetos que comienzan por "botón":

```
OBJECT GET COORDINATES(*;"botón@";izquierda;superior;derecha;inferior)
```

Ejemplo 2

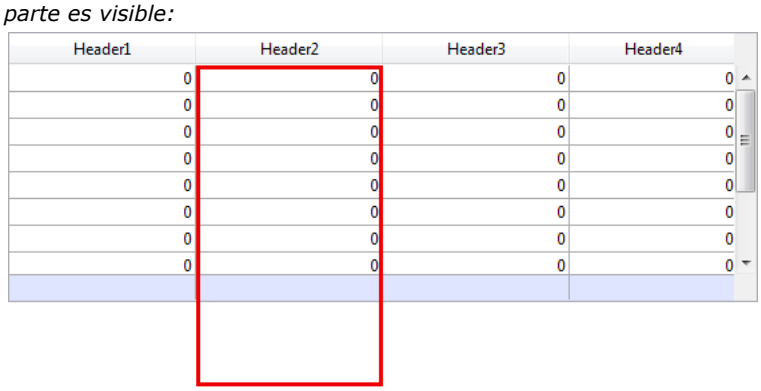
Para las necesidades de su interfaz, usted desea rodear el área en la que el usuario hizo clic con un rectángulo rojo:



En el método objeto del listbox, puede escribir:

```
OBJECT SET VISIBLE(*;"rectangleInfo";False) //inicializar un rectángulo rojo
$ptr:=OBJECT Get pointer(Object current)
OBJECT GET COORDINATES($ptr->;$x1;$y1;$x2;$y2)
OBJECT SET VISIBLE(*;"RedRect";True)
OBJECT SET COORDINATES(*;"RedRect";$x1;$y1;$x2;$y2)

OBJECT GET COORDINATES(*;"LB1";$lbx1;$lby1;$lbx2;$lby2)
If($lby1>$y1)|($lby2<$y2)  // si el área en la que se hace clic está fuera del listbox
   OBJECT SET VISIBLE(*;"Alert";True)  // mostrar un alerta
Else
   OBJECT SET VISIBLE(*;"Alert";False)
End if
```

El método devuelve las coordenadas teóricas. Si el listbox es redimensionado, puede necesitar calcular el recorte para saber qué parte es visible:



OBJECT Get corner radius

OBJECT Get corner radius ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
Resultado	Entero largo	⮌	Radio de esquinas redondeadas (en píxeles)

Descripción

El comando ***OBJECT Get corner radius*** *devuelve el valor actual del radio de la esquina para el objeto rectángulo redondeado cuyo nombre se pasa en el parámetro objeto. Este valor puede haber sido definido a nivel del formulario utilizando la lista de propiedades (ver* ***Radio de la esquina (rectángulos)****), o para el proceso actual con el comando* ***OBJECT SET CORNER RADIUS****.*

*Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

Nota: *en la versión actual de 4D, este comando sólo se aplica a los rectángulos redondeados (que son objetos estáticos). Como resultado, sólo la sintaxis basada en el nombre de objeto (usando el parámetro *) es compatible.*

Este comando devuelve el radio de esquinas redondeadas en píxeles. Por defecto, este valor es de 5 píxeles.

Ejemplo

El siguiente código puede añadirse a un método de un botón:

```
C_LONGINT($radius)
$radius:=OBJECT Get corner radius(*;"GreenRect") //obtiene el valor actual
OBJECT SET CORNER RADIUS(*;"GreenRect";$radius+1) //aumenta el radio
 // El valor máximo se manejará automáticamente: cuando se alcance, el botón
 // no hará nada
```

OBJECT Get data source

OBJECT Get data source ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
Resultado	Puntero	⮌	Puntero a la fuente de datos actual del objeto

Descripción

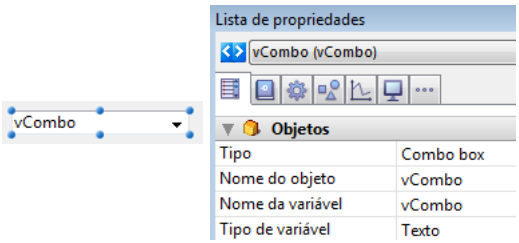
El comando ***OBJECT Get data source*** *devuelve la fuente de datos actual de los objetos designados por los parámetros objeto y * .*

Puede definir la fuente de datos para un objeto en modo Diseño utilizando la Lista de propiedades, o utilizando el comando ***OBJECT SET DATA SOURCE****.*

*Pasando el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

Ejemplo

Dado un objeto combo box definido en un formulario:



Ejecuta el siguiente código:

```
$vPtr :=OBJECT Get data source(*;"vCombo")
 // $vPtr contiene -> vCombo
```

OBJECT GET DRAG AND DROP OPTIONS

OBJECT GET DRAG AND DROP OPTIONS ({* ;} objeto ; arrastrable ; arrastrableAuto ; soltable ; soltableAuto)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
arrastrable	Booleano	⬅	0 = False, 1 = True
arrastrableAuto	Booleano	⬅	0 = False, 1 = True
soltable	Booleano	⬅	0 = False, 1 = True
soltableAuto	Booleano	⬅	0 = False, 1 = True

Descripción

*El comando **OBJECT GET DRAG AND DROP OPTIONS** devuelve las opciones de arrastrar y soltar para el objeto o los objetos designados por los parámetros objeto y * para el proceso actual.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

*El comando devuelve las opciones de arrastrar y soltar actuales, como están definidas en modo Diseño o para el proceso actual utilizando el comando **OBJECT SET DRAG AND DROP OPTIONS**.*

Cada parámetro devuelve True o False dependiendo de si la opción correspondiente está activa o inactiva:

- *arrastrable = True: objeto arrastrable en modo programado.*
- *arrastrableAuto = True (utilizable únicamente con los campos y variables texto, combo boxes y list boxes): Objeto arrastrable en modo automático.*
- *soltable = True: objeto acepta soltar en modo programado.*
- *soltableAuto = True (utilizable únicamente con los campos y variables imagen, texto, combo boxes y list boxes): Objeto acepta soltar en modo automático.*

OBJECT Get enabled

OBJECT Get enabled ({* ;} objeto) -> Resultado

Parámetro	*Tipo*		*Descripción*
*	*Operador*	→	*Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo*
objeto	*Objeto de formulario*	→	*Nombre del objeto (si se especifica *) o Variables (si se omite *)*
Resultado	*Booleano*	←	*True = objeto(s) activo(s), de lo contrario False*

Descripción

*El comando **OBJECT Get enabled** devuelve True si el objeto o grupo de objetos designado por objeto está activo en el formulario y False si no está activo.*

Un objeto activo reacciona a los clics y atajos de teclado.

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable (variable objeto únicamente) en lugar de una cadena.*

Este comando se puede aplicar a los siguientes tipos de objetos:

- *Botón, Botón por defecto, Botón 3D, Botón invisible, Botón inverso*
- *Botón radio, Botón radio 3D, Botón Imagen*
- *Casilla de selección, Casilla de selección 3D*
- *Pop-up menú, Lista desplegable, Combo box, Menú/Lista desplegable*
- *Termómetro, Regla*

OBJECT Get enterable

OBJECT Get enterable ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	→	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	→	Nombre del objeto (si se especifica *) o variable o campo (si se omite *)
Resultado	Booleano	←	True = objeto(s) editable(s)

Descripción

El comando ***OBJECT Get enterable*** *devuelve True si el objeto o grupo de objetos designado por objeto tiene el atributo* ***editable****; de lo contrario, devuelve False.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

OBJECT GET EVENTS

OBJECT GET EVENTS ({* ;} objeto ; arrEvents)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto "" para designar el formulario (si se especifica *) o Campo o variable (si se omite *)
arrEvents	Array entero largo	⬅	Array de eventos desactivados

Descripción

El comando ***OBJECT GET EVENTS*** *permite obtener la configuración actual de los eventos formulario del formulario, del objeto o de los objetos designado(s) por los parámetros objeto y *.*

Los eventos formulario se pueden activar/desactivar, ya sea utilizando la lista de propiedades o utilizando el comando ***OBJECT SET EVENTS*** *si se le llama en el proceso actual.*

*Si se pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*
*Para obtener la configuración de los eventos del formulario para el propio formulario, pase el parámetro opcional * y una cadena vacía "" en objeto: en este caso, usted designa el formulario actual.*

Nota: *si desea obtener los eventos de un subformulario relacionados con una tabla, sólo se puede utilizar la sintaxis basada en el nombre del objeto.*

*Pase un array entero largo en el parámetro arrEvents. Cuando se ejecuta el comando, se asigna el tamaño a este array automáticamente y recibe todos los eventos formulario predefinidos o personalizados que se han activado para el objeto o el formulario. Puede comparar los valores recibidos con las constantes del tema "****Eventos de formulario****".*

Tenga en cuenta que el array arrEvents se devuelve vacío si ningún método objeto está asociado al objeto o si ningún método formulario se asocia al formulario.

Ejemplo

Usted quiere activar dos eventos y obtener la lista de eventos para un objeto:

```
ARRAY LONGINT($ArrCurrentEvents;0)
ARRAY LONGINT($ArrEnabled;2)
$ArrEnabled{1}:=On Header Click
$ArrEnabled{2}:=On Footer Click
OBJECT SET EVENTS(*;"Col1";$ArrEnabled;Enable events others unchanged)
OBJECT GET EVENTS(*;"Col1";$ArrCurrentEvents)
```

OBJECT Get filter

OBJECT Get filter ({* ;} objeto) -> Resultado

Parámetro	*Tipo*		*Descripción*
*	*Operador*	➡	*Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo*
objeto	*Objeto de formulario*	➡	*Nombre del objeto (si se especifica *) o variable o campo (si se omite)*
Resultado	*Texto*	➡	*Nombre del filtro*

Descripción

*El comando **OBJECT Get filter** devuelve el nombre de todo filtro asociado con el objeto o grupo de objetos designado por objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

OBJECT Get focus rectangle invisible

OBJECT Get focus rectangle invisible ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable o campo (si se omite *)
Resultado	Booleano	⮌	True = rectángulo de foco oculto, False = rectángulo de foco visible

Descripción

El comando ***OBJECT Get focus rectangle invisible*** *devuelve el estado de la opción de invisibilidad del rectángulo de foco del objeto o de los objetos designados por los parámetros objeto y * para el proceso actual. Esta configuración corresponde a la opción* ***Ocultar rectángulo de foco*** *disponible para los objetos editables en la Lista de propiedades en modo Diseño. Este comando devuelve el estado actual de la opción, como se definió en modo Diseño o utilizando el comando* ***OBJECT SET FOCUS RECTANGLE INVISIBLE****.*

Nota: *puede utilizar esta opción únicamente en Mac OS. No tiene efecto en Windows.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable o un campo. En este caso, se pasa una referencia de variable en lugar de una cadena.*

El comando devuelve ***True*** *si el rectángulo de foco está oculto y* ***False*** *cuando es visible.*

OBJECT Get font

OBJECT Get font ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
Resultado	Texto	⮌	Nombre de la fuente

Descripción

*El comando **OBJECT Get font** devuelve el nombre de la fuente utilizada por el objeto de formulario designado por objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

OBJECT Get font size

OBJECT Get font size ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o variable o campo (si se omite *)
Resultado	Entero largo	⮌	Tamaño de la fuente en puntos

Descripción

*El comando **OBJECT Get font size** devuelve el tamaño (en puntos) de la fuente utilizada por el objeto de formulario designado por objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

OBJECT Get font style

OBJECT Get font style (* ; objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
Resultado	Entero largo	⮌	Estilo de la fuente

Descripción

*El comando **OBJECT Get font style** devuelve el estilo actual de la fuente utilizada por el objeto de formulario designado por objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

*Puede comparar el valor devuelto por el comando con el valor de uno o más de las siguientes constantes predefinidas, ubicadas en el tema "**Estilos de fuente**":*

Constante	Tipo	Valor
Plain	Entero largo	0
Bold	Entero largo	1
Italic	Entero largo	2
Underline	Entero largo	4

OBJECT Get format

OBJECT Get format ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
Resultado	Cadena	⮌	Formato de salida del objeto

Descripción

*El comando **OBJECT Get format** devuelve el formato de salida actual aplicado al objeto especificado en el parámetro objeto.*

*Si pasa el parámetro opcional *, indica un nombre de objeto (en este caso, pase una cadena en objeto). Si omite este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, usted no pasa una cadena, sino una referencia de campo o de variable.*

*Este comando devuelve el formato de salida actual del objeto, es decir el formato definido en el entorno Diseño o utilizando el comando **OBJECT SET FORMAT**. **OBJECT Get format** funciona con todos los tipos de objetos de formulario (campos o variables) que aceptan un formato de salida: booleano, fecha, hora, imagen, cadena, numérico, como también con rejillas de botones, dials, termómetros, reglas, menús imagen desplegable, botones imagen, botones 3D y encabezados de list box. Para mayor información sobre formatos de salida de estos objetos, consulte la documentación del comando **OBJECT SET FORMAT**.*

***Nota:** si aplica el comando a un conjunto de objetos, el formulario, se devuelve el formulario del último objeto seleccionado.*

*Cuando el comando **OBJECT Get format** se aplica a objetos de tipo fecha, hora o imagen (formatos definidos como constantes), la cadena que se devuelve corresponde al código del carácter de la constante. Para obtener el valor de la constante, simplemente aplique la función **Character code** al resultado (ver ejemplo a continuación).*

Ejemplo 1

Este ejemplo le permite obtener el valor de la constante del formato aplicado a la variable imagen llamada "mifoto":

```
C_STRING(2;$formato)
OBJECT SET FORMAT(*;"mifoto";Char(On background))
  `Aplicación del formato de fondo (valor = 3)
$formato:=OBJECT Get format(*;"mifoto")
ALERT("Formato número:"+String(Character code($formato)))
  `Mostrar el valor "3"
```

Ejemplo 2

Este ejemplo le permite obtener el formato aplicado al campo booleano [Miembros]Estado_civil:

```
C_STRING(30;$formato)
$formato:=OBJECT Get format([Miembros]Estado_civil)
ALERT($formato)  `Visualizar formato, por ejemplo "Casado;Soltero"
```

OBJECT Get help tip

OBJECT Get help tip ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Texto	⮌	Mensaje de ayuda del objeto

Descripción

El comando ***OBJECT Get help tip*** *devuelve el mensaje de ayuda asociado al objeto o a los objetos designados por los parámetros objeto y * en el proceso actual.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

El comando devuelve el mensaje de ayuda actual asociado al objeto, tal como está definido en el modo Diseño o para el proceso utilizando el comando ***OBJECT SET HELP TIP****. La cadena devuelta muestra el mensaje como aparece cuando se ejecuta el formulario. Si contiene elementos variables (resname xliff o referencias 4D), se interpretan en función del contexto.*

Ejemplo

El título de un botón imagen se guarda en forma de mensaje de ayuda. Este título se guarda en un archivo xliff y difiere en función del lenguaje actual de la aplicación:

```
OBJECT SET HELP TIP(*;"button1";":xliff:btn_discover")
$helpmessage:=OBJECT Get help tip(*;"button1")
 // $helpmessage contiene por ejemplo "Découvrir" con un 4D francés y "Discover" con un 4D inglés.
```

OBJECT Get horizontal alignment

OBJECT Get horizontal alignment ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre del objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
Resultado	Entero largo	⮌	Código de alineación

Descripción

El comando ***OBJECT Get horizontal alignment*** *devuelve un código indicando el tipo de alineación horizontal aplicada al objeto designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto designa el nombre de un objeto (una cadena). Si no pasa el parámetro *, indica que el parámetro objeto designa un campo o una variable. En este caso, usted no pasa una cadena sino la referencia de un campo o de una variable (campo o variable de tipo objeto únicamente).*

Nota: *si aplica el comando a un grupo de objetos, sólo devuelve el valor de alineación del último objeto.*

El código devuelto corresponde a una de las siguientes constantes, ubicadas en el tema ***Propiedades de los objetos****:*

Constante	Tipo	Valor	Comentario
Align center	Entero largo	3	
Align default	Entero largo	1	
Align left	Entero largo	2	
Align right	Entero largo	4	
wk justify	Entero largo	5	Disponible para áreas 4D Write Pro únicamente

Nota: *la constante wk justify está disponible en el tema "****4D Write Pro****".*

Los objetos de formulario a los cuales se puede aplicar alineación son los siguientes:

- *Áreas de desplazamiento*
- *Combo box*
- *Textos estáticos*
- *Áreas de grupos*
- *Menús desplegables/Listas desplegables*
- *Campos*
- *Variables*
- *List boxes*
- *Columnas de list box*
- *Encabezados de list box*
- *Pies de list box*
- *Áreas* ***Referencia 4D Write Pro***

OBJECT Get indicator type

OBJECT Get indicator type ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Entero largo	⬅	Tipo de indicador

Descripción

El comando ***OBJECT Get indicator type*** *devuelve el tipo del indicador actual asignado al termómetro(s) designado por los parámetros objeto y *.*

Puede definir el tipo de indicador utilizando la lista de propiedades en modo de diseño o utilizando el comando ***OBJECT SET INDICATOR TYPE****.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable del objeto únicamente).*

*Puede comparar el valor devuelto por el comando con las siguientes constantes, que se encuentran en el tema "****Propiedades de los objetos****":*

Constante	Tipo	Valor	Comentario
Asynchronous progress bar	Entero largo	3	Indicador circular muestra una animación continua
Barber shop	Entero largo	2	Barra que muestra una animación continua
Progress bar	Entero largo	1	Barra de progreso estándar

OBJECT Get keyboard layout

OBJECT Get keyboard layout ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable o campo (si se omite *)
Resultado	Cadena	⮌	Código del lenguaje de configuración, "" = sin configuración

Descripción

El comando ***OBJECT Get keyboard layout*** *devuelve la configuración del teclado actual asociada a los objetos designados por los parámetros objeto y * para el proceso actual.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable o un campo. En este caso, pase una referencia en lugar de un nombre.*

El comando devuelve una cadena indicando el código del lenguaje utilizado, basado en RFC3066, ISO639 e ISO3166. Para mayor información, consulte la descripción del comando ***SET DATABASE LOCALIZATION****.*

OBJECT Get list name

OBJECT Get list name ({* ;} objeto {; tipoLista}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre del objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
tipoLista	Entero largo	➡	Tipo de lista: lista de selección, lista de obligatorios o lista de excluidos
Resultado	Texto	⬅	Nombre de la lista (definida en modo Diseño)

Descripción

El comando ***OBJECT Get list name*** *devuelve el nombre de la lista asociada al objeto o a un grupo de objetos designados por objeto. 4D le permite asociar una lista de opciones (creada con el editor de la listas en modo Diseño) a los objetos de formulario utilizando el editor de formularios o el comando* ***OBJECT SET LIST BY NAME****.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o de variable (campo o variable objeto únicamente) en lugar de una cadena.*

*Puede utilizar el parámetro opcional tipoLista para designar el tipo de lista que desea obtener. Por defecto, si omite este parámetro, el comando devuelve el nombre de la lista de opciones (lista de valores) asociado al objeto. También puede obtener los nombres de las listas obligatorias o de las listas de excluidos pasando, en tipoLista, una de las siguientes constantes que se encuentran en el tema "****Propiedades de los objetos****":*

Constante	Tipo	Valor	Comentario
Choice list	Entero largo	0	Lista simple de selección de valores (opción "Lista" en la Lista de Propiedades) (por defecto)
Excluded list	Entero largo	2	Lista de valores no aceptados para la entrada (Opción "Exclusiones" en la lista de propiedades)
Required list	Entero largo	1	Lista sólo los valores aceptados para la entrada (Opción "Obligatoria" en la Lista de Propiedades)

Si ninguna lista del tipo definido está asociada al objeto, el comando devuelve una cadena vacía ("").

OBJECT Get list reference

OBJECT Get list reference ({* ;} objeto {; tipoLista}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
tipoLista	Entero largo	➡	Tipo de lista: lista de Selección, lista obligatoria o la lista de excluidos
Resultado	ListRef	⮌	Número de referencia de lista

Descripción

El comando ***OBJECT Get list reference*** *devuelve el número de referencia (RefList) de la lista jerárquica asociada al objeto o grupo de objetos designados por los parámetros objeto y * .*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*Por defecto, si omite el parámetro tipoLista, el comando devuelve el nombre de la lista de opciones (lista de valores) asociado al objeto. También puede obtener el número de referencia de las listas obligatoria o las listas de excluidos pasando en tipoLista, una de las siguientes constantes del tema "**Propiedades de los objetos**":*

Constante	Tipo	Valor	Comentario
Choice list	Entero largo	0	Lista simple de selección de valores (opción "Lista" en la Lista de Propiedades) (por defecto)
Excluded list	Entero largo	2	Lista de valores no aceptados para la entrada (Opción "Exclusiones" en la lista de propiedades)
Required list	Entero largo	1	Lista sólo los valores aceptados para la entrada (Opción "Obligatoria" en la Lista de Propiedades)

Si no hay una lista jerárquica asociada al objeto para el tipoLista definido, el comando devuelve 0.

OBJECT GET MAXIMUM VALUE

OBJECT GET MAXIMUM VALUE ({* ;} objeto ; valorMax)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
valorMax	Fecha, Hora, Número	⬅	Valor máximo actual para objeto

Descripción

El comando ***OBJECT GET MAXIMUM VALUE*** *devuelve, en la variable valorMax, el valor máximo actual del objeto o de los objetos designados por los parámetros objeto y * .*

Puede establecer la propiedad "Valor máximo" con la lista de propiedades en modo Diseño o utilizando el comando ***OBJECT SET MAXIMUM VALUE****.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

.

OBJECT GET MINIMUM VALUE

OBJECT GET MINIMUM VALUE ({* ;} objeto ; valorMin)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
valorMin	Fecha, Hora, Número	⬅	Valor mínimo actual del objeto

Descripción

El comando ***OBJECT GET MINIMUM VALUE*** *devuelve, en la variable* valorMin*, el valor mínimo actual del objeto o de los objetos designado(s) por los parámetros* objeto *y* *.

La propiedad "Valor mínimo" puede definirse utilizando la Lista de propiedades en modo Diseño, o utilizando el comando ***OBJECT SET MINIMUM VALUE****.*

Si pasa el parámetro opcional * *indica que el parámetro* objeto *es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro* objeto *es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

OBJECT Get multiline

OBJECT Get multiline ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
Resultado	Entero largo	⮌	Estado Multilínea del objeto

Descripción

El comando ***OBJECT Get multiline*** *devuelve el estado actual de la opción "Multilínea" del objeto o de los objetos designado(s) por los parámetros objeto y * .*

Puede definir la opción "Multilínea" de un objeto utilizando la lista de propiedades o utilizando el comando ***OBJECT SET MULTILINE****.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*El valor devuelto corresponde a una de las siguientes constantes del tema "****Propiedades de los objetos****":*

Constante	Tipo	Valor	Comentario
Multiline Auto	Entero largo	0	En las áreas de una sola línea, las palabras situadas al final de las líneas se cortan y no hay retornos de línea. En las áreas de varias líneas, 4D efectúa saltos de línea automáticos.
Multiline No	Entero largo	2	Nunca hay vuelta de la línea: el texto se muestra siempre en una sola línea. Si el campo o la variable alfa o texto contiene retornos de carro, el texto situado después del primer retorno de carro se elimina tan pronto como se modifica el área.
Multiline Yes	Entero largo	1	En las áreas de una sola línea, el texto se muestra hasta el primer retorno de carro o hasta la última palabra que se puede mostrar por completo. 4D inserta retornos de línea, es posible desplazarse por el contenido del área con la tecla de flecha hacia abajo. En las áreas de varias líneas, 4D efectúa los saltos de línea automáticos.

Nota: *si aplica el comando* ***OBJECT Get multiline*** *a un objeto que no admite la opción "Multilínea", devuelve el valor 0.*

OBJECT Get name

OBJECT Get name {(selector)} -> Resultado

Parámetro	Tipo		Descripción
selector	Entero largo	→	Categoría de objeto
Resultado	Texto	←	Nombre del objeto

Descripción

*El comando **OBJECT Get name** devuelve el nombre de un objeto de formulario.*
El comando permite designar dos tipos de objetos en función del parámetro selector. En este parámetro, puede pasar una de las siguientes constantes (ubicadas en el tema ""):

- *Object current o selector omitido: si pasa este selector u omite el parámetro selector, el comando devuelve el nombre del objeto a partir del cual fue llamado (método objeto o submétodo llamado por el método de objeto). En este caso, el comando debe llamarse en el contexto de un objeto de formulario, de lo contrario devuelve una cadena vacía.*
- *Object with focus: si pasa este selector, el comando devuelve el nombre del objeto que tiene el foco en el formulario.*

Ejemplo

Método objeto del botón "bValidateForm":

```
$btnName:=OBJECT Get name(Object current)
```

Después de la ejecución de este método objeto, la variable $btnName contiene el valor "bValidateForm".

OBJECT Get placeholder

OBJECT Get placeholder ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
Resultado	Texto	⮌	Texto de ejemplo asociado al objeto

Descripción

El comando ***OBJECT Get placeholder*** *devuelve el texto de ejemplo asociado al objeto o a los objetos designado(s) por los parámetros objeto y *. Si no hay ningún texto del marcador asociado con el objeto, el comando devuelve una cadena vacía.*

Puede definir el texto del marcador, ya sea usando la lista de propiedades o utilizando el comando ***OBJECT SET PLACEHOLDER****.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable de objeto únicamente).*

Si el marcador es una referencia xliff definida por la lista de propiedades, el comando devuelve la referencia original en el formulario ":xliff:resname", y no a su valor calculado.

Ejemplo

Si quiere recibir el texto del marcador de campo:

```
$txt:=OBJECT Get placeholder([People]LastName)
```

OBJECT Get pointer

OBJECT Get pointer {(selector {; nomObjeto {; nomSubFormulario}})} -> Resultado

Parámetro	Tipo		Descripción
selector	Entero largo	➡	Categoría del objeto
nomObjeto	Texto	➡	Nombre de objeto
nomSubFormulario	Texto	➡	Nombre del objeto subformulario
Resultado	Puntero	⬅	Puntero a la variable del objeto

Descripción

*El comando **OBJECT Get pointer** devuelve un puntero a la variable de un objeto de formulario.*

*Este comando se puede utilizar para designar diferentes tipos de objetos en función del valor del parámetro selector. Puede pasar en este parámetro una de las siguientes constantes (del tema **Objetos de formulario (Acceso)**):*

- *Object current o selector omitido: si se omite el parámetro selector o pasar este selector, el comando devuelve un puntero a la variable asociada al objeto actual (objeto cuyo método está en ejecución).*
  ***Nota**: este funcionamiento es estrictamente equivalente al funcionamiento previo del comando **Self**. El comando **Self** se conserva únicamente por razones de compatibilidad.*
- *Object with focus: si pasa este selector, el comando devuelve un puntero a la variable asociada al objeto que tiene el foco en el formulario. Los últimos dos parámetros opcionales se ignoran si se pasan.*
  ***Nota**: este funcionamiento es estrictamente equivalente al comando **Focus object**. El comando **Focus object** ahora es obsoleto a partir de 4D v12.*
- *Object subform container: si pasa este selector, el comando devuelve un puntero a la variable vinculada con el contenedor del subformulario. Los últimos dos parámetros opcionales se ignoran si se pasan. Este selector por lo tanto sólo puede utilizarse en el contexto de un formulario que se utiliza como un subformulario, con el fin de acceder a la variable asociada al objeto contenedor.*
- *Object named: si pasa este selector, también debe pasar el segundo parámetro, nomObjeto. En este caso, el comando devuelve un puntero a la variable asociada al objeto cuyo nombre se pasó en este parámetro.*
  ***Nota**: si nomObjeto corresponde a un subformulario y la opción "Subformulario salida" está seleccionada, el comando devuelve un puntero a la tabla del subformulario si una tabla fuente está especificada, de lo contrario devuelve Nil.*

***Nota:** When it is used in the context of a list box, **OBJECT Get pointer** with Object current or Object with focus selectors returns a pointer to the list box, the column, or the header, depending on the context. For more information, please refer to the **Gestión programada de los objetos de tipo List box** page.*

El parámetro opcional nomSubForm permite recuperar un puntero a un objeto nomObjeto que no pertenece al contexto actual, es decir, al formulario padre. Para poder utilizar este parámetro, debe haber pasado el selector Object named.
*Cuando se pasa el parámetro nomSubForm, el comando **OBJECT Get pointer** primero busca el objeto subformulario objeto llamado nomSubForm en el formulario actual, luego busca al interior de este subformulario un objeto llamado nomObjeto. Si este objeto se encuentra, se devuelve un puntero a la variable de este objeto.*

Ejemplo

Dado un formulario "SF" utilizado dos veces como subformulario en el mismo formulario padre. Los objetos subformularios se llaman "SF1" y "SF2". El formulario "SF" contiene un objeto llamado ValorActual. En el evento "On Load" del método de formulario del formulario padre, queremos inicializar el objeto Valor Actual de SF1 en "Enero" y el de SF2 en "Febrero":

```
C_POINTER($Ptr)
$Ptr:=OBJECT Get pointer(Object named;"Valor actual";"SF1")
$Ptr->:="Enero"
$Ptr:=OBJECT Get pointer(Object named;"Valor actual";"SF2")
$Ptr->:="Febrero"
```

OBJECT GET PRINT VARIABLE FRAME

OBJECT GET PRINT VARIABLE FRAME ({* ;} objeto ; tamVariable {; subformFijo})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
tamVariable	Booleano	⬅	True = Impresión tamaño variable, False = impresión tamaño fijo
subformFijo	Entero largo	⬅	Opción para impresión de subformularios de tamaño fijo

Descripción

El comando ***OBJECT GET PRINT VARIABLE FRAME*** *obtiene la configuración actual de las opciones de impresión en tamaño variable del objeto o de los objetos designado(s) por los parámetros objeto y * .*

Las propiedades de impresión de tamaño variable se pueden definir utilizando la lista de propiedades o el comando ***OBJECT SET PRINT VARIABLE FRAME****.*
*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*En el parámetro tamVariable, el comando devuelve una variable booleana cuyo valor indica el estado activo (****True****) o inactivo (****False****) de la impresión en tamaño variable.*

*Si el objeto es un subformulario y si la impresión en tamaño variable está desactivada (****False****), el comando también devuelve en el parámetro subformFijo, la opción de impresión en tamaño fijo del subformulario. Puede comparar el valor devuelto con las siguientes constantes, del tema "****Propiedades de los objetos****":*

Constante	Tipo	Valor	Comentario
Print Frame fixed with multiple records	Entero largo	2	El tamaño inicial del marco permanece del mismo tamaño, 4D imprime el formulario varias veces para incluir todos los registros.
Print Frame fixed with truncation	Entero largo	1	4D imprime sólo los registros que aparecen en el área del subformulario. El formulario se imprime sólo una vez y los registros que no se imprimen se ignoran.

OBJECT GET RESIZING OPTIONS

OBJECT GET RESIZING OPTIONS ({* ;} objeto ; horizontal ; vertical)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
horizontal	Entero largo	⬅	Opción de redimensionamiento horizontal
vertical	Entero largo	⬅	Opción de redimensionamiento vertical

Descripción

El comando ***OBJECT GET RESIZING OPTIONS*** *devuelve las opciones de redimensionamiento actuales del o de los objetos designados por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, pase una referencia de variable en lugar de una cadena.*

El comando devuelve las opciones de redimensionamiento actuales, como se definieron en el modo Diseño o para el proceso utilizando el comando ***OBJECT SET RESIZING OPTIONS****. Estas opciones definen la visualización del objeto en caso de redimensionamiento de la ventana del formulario.*

El parámetro horizontal devuelve un valor indicando la opción de redimensionamiento horizontal definido para el objeto. Puede comparar el valor recibido con las siguientes constantes, del tema ***Propiedades de los objetos****:*

Constante	Tipo	Valor	Comentario
Resize horizontal grow	Entero largo	1	Si la ventana se agranda un 50% de ancho, el objeto se agranda 50% a la derecha
Resize horizontal move	Entero largo	2	Si la ventana se agranda 100 píxeles de ancho, el objeto se mueve 100 píxeles a la derecha
Resize horizontal none	Entero largo	0	Si la ventana se agranda de ancho, ni el largo ni la posición del objeto varían

El parámetro vertical devuelve un valor indicando la opción de redimensionamiento vertical definido para el objeto. Puede comparar el valor recibido con las siguientes constantes, del tema ***Propiedades de los objetos****:*

Constante	Tipo	Valor	Comentario
Resize vertical grow	Entero largo	1	Si la ventana se agranda un 50% de alto, el objeto se alarga 50% hacia abajo
Resize vertical move	Entero largo	2	Si la ventana se agranda 100 píxeles de alto, el objeto se mueve 100 píxeles hacia abajo
Resize vertical none	Entero largo	0	Si la ventana se agranda de alto, ni el ancho ni la posición del objeto varían

OBJECT GET RGB COLORS

OBJECT GET RGB COLORS ({* ;} objeto ; colorPrimerPlano {; colorFondo {; colorFondoAlt}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o variable o campo (si se omite *)
colorPrimerPlano	Entero largo	⬅	Valor del color RGB del primer plano
colorFondo	Entero largo	⬅	Valor del color RGB del fondo
colorFondoAlt	Entero largo	⬅	Valor del color RGB del fondo alterno

Descripción

El comando ***OBJECT GET RGB COLORS*** *devuelve los colores de fondo y primer plano del objeto o grupo de objetos designados por objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

Cuando el comando se aplica a un objeto de tipo List box, el color de fondo alterno de las filas pueden se devuelto en el parámetro colorFondoAlt. En este caso, el valor de colorFondo se utiliza para el fondo de las líneas impares.

Los valores de colores RGB devueltos en los parámetros a colorPrimerPlano, colorFondo y colorFondoAlt son enteros largos de 4 bytes en formato (0x00RRGGBB) o los valores negativos corresponden a los colores "sistema". En este último caso, puede comparar el valor obtenido con las constantes del tema ***DEFINIR COLORES RVA****:*

Constante	Tipo	Valor	Comentario
Background color	Entero largo	-2	
Background color none	Entero largo	-16	Esta constante puede ser utilizada únicamente con los parámetros colorFondo y colorFondoAlt.
Dark shadow color	Entero largo	-3	
Disable highlight item color	Entero largo	-11	
Foreground color	Entero largo	-1	
Highlight menu background color	Entero largo	-9	
Highlight menu text color	Entero largo	-10	
Highlight text background color	Entero largo	-7	
Highlight text color	Entero largo	-8	
Light shadow color	Entero largo	-4	

Nota: *los colores "sistema" se aplican utilizando el comando* ***OBJECT SET RGB COLORS****.*

Para obtener más información acerca del formato de los parámetros colorPrimerPlano, colorFondo y colorFondoAlt, consulte la descripción del comando ***OBJECT SET RGB COLORS****.*

OBJECT GET SCROLL POSITION

OBJECT GET SCROLL POSITION ({* ;} objeto ; posicionLinea {; posicionH})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable o campo (si se omite *)
posicionLinea	Entero largo	⬅	Número de la primera línea mostrada o desplazamiento vertical en píxeles (imágenes)
posicionH	Entero largo	⬅	Número de la primera columna mostrada (list box) o desplazamiento horizontal en píxeles (imágenes)

Descripción

OBJECT GET SCROLL POSITION devuelve, en los parámetros posicionLinea y posicionH, información relacionada con la posición de las barras de desplazamiento del objeto de formulario designado por los parámetros * y objeto.

Si pasa el parámetro opcional *, indica que el parámetro objeto es el nombre de un objeto de tipo subformulario, lista jerárquica, área de desplazamiento, list box o imagen (en este caso, pase una cadena en objeto). Si no pasa este parámetro, indica que el parámetro objeto es una variable (RefLista de lista jerárquica, imagen o list box) o un campo.

Nota: con los objetos de tipo subformulario, sólo se soporta la sintaxis con *.

Si objeto designa un objeto de tipo lista (subformulario, lista jerárquica, área de desplazamiento o list box), posicionLinea devuelve el número de la primera línea mostrada en el objeto. posicionH (list box únicamente) devuelve el número de la primera columna mostrada en la parte izquierda del list box. Con otros tipos de objetos, este parámetro devuelve 0.

Si objeto designa una imagen (variable o campo), posicionLinea devuelve el desplazamiento vertical y posicionH el desplazamiento horizontal de la imagen. Estos valores se expresan en píxeles con respecto al origen de la imagen en el sistema de coordenadas locales.

OBJECT GET SCROLLBAR

OBJECT GET SCROLLBAR ({* ;} objeto ; horizontal ; vertical)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o variable o campo (si se omite*)
horizontal	Booleano, Entero largo	⬅	True = mostrado, False = oculto
vertical	Booleano, Entero largo	⬅	True = mostrado, False = oculto

Descripción

*El comando **OBJECT GET SCROLLBAR** se utiliza para mostrar u ocultar las barras de desplazamiento horizontal y/o vertical en el objeto designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

En los parámetros horizontal y vertical, puede pasar variables de tipo Booleano o entero largo:

- *Cuando pase variables Booleanas, el valor devuelto refleja el estado **actual** de la barra de desplazamiento:*
  - *Si la barra de desplazamiento se ha definido como oculta, el parámetro recibe False,*
  - *Si la barra de desplazamiento se ha definido como visible, el parámetro recibe True,*
  - *Si la barra de desplazamiento se ha definido en modo automático, el parámetro recibe True o False dependiendo del estado de visualización actual del objeto.*
- *Cuando pase variables enteras largas, el valor devuelto refleja la visibilidad definida para la barra de desplazamiento:*
  - *Si la barra de desplazamiento se ha definido como oculta, el parámetro recibe 0,*
  - *Si la barra de desplazamiento se ha definido como visible, el parámetro recibe 1,*
  - *Si la barra de desplazamiento se ha definido en modo automático, el parámetro recibe 2.*

Este comando se puede utilizar con los siguientes objetos de formulario:

- *campos y variables objeto texto o imagen*
- *list boxes,*
- *listas jerárquicas,*
- *subformularios.*

*Para obtener más información, consulte la descripción del comando **OBJECT SET SCROLLBAR**.*

OBJECT GET SHORTCUT

OBJECT GET SHORTCUT ({* ;} objeto ; tecla ; modificadores)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
tecla	Cadena	⬅	Tecla asociada al objeto
modificadores	Entero largo	⬅	Máscara o combinación de máscaras de teclas de modificación

Descripción

*El comando **OBJECT GET SHORTCUT** devuelve el atajo de teclado asociado al objeto o a los objetos designados por los parámetros objeto y * en el proceso actual.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

*El parámetro tecla devuelve el carácter asociado a la tecla (en el caso de una tecla estándar) o una cadena entre corchetes que designa la tecla (en el caso de una tecla función). Puede comparar este valor con las constantes del tema **Atajos de teclado** (ver el comando **OBJECT SET SHORTCUT**).*

*El parámetro modificadores devuelve un valor que indica la(s) tecla(s) modificadora(s) asociada(s) al atajo. Si hay varias teclas modificadoras combinadas, el comando devuelve la suma de sus valores. Puede comparar el valor recibido con las siguientes constantes, del tema **Eventos (Modificadores)**:*

Constante	Tipo	Valor	Comentario
Command key mask	Entero largo	256	Tecla Ctrl en Windows, Tecla Comando en OS X
Control key mask	Entero largo	4096	Tecla Ctrl bajo OS X, o clic derecho en Windows y OS X
Option key mask	Entero largo	2048	Tecla Alt (también llamada Opción en OS X)
Shift key mask	Entero largo	512	Windows y OS X

Si ninguna tecla de modificación se ha definido para el atajo, modificadores devuelve 0.

***Nota**: si el parámetro objeto designa varios objetos del formulario que tienen diferentes configuraciones, el comando devuelve "" en tecla 0 en modificadores.*

OBJECT Get style sheet

OBJECT Get style sheet ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
Resultado	Texto	⮌	Nombre de la hoja de estilo

Descripción

El comando ***OBJECT Get style sheet*** *devuelve el nombre de la hoja de estilos asociada al objeto o a los objetos designado(s) por los parámetros objeto y * .*

La hoja de estilo puede haber sido asignada en modo Diseño utilizando la lista de propiedades o para el proceso actual usando el comando ***OBJECT SET STYLE SHEET****.*

*Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

El comando puede devolver:

- *un nombre de hoja de estilos,*
- *una cadena vacía ("") si no se asigna ninguna hoja de estilos o*
- *si una hoja de estilo "Automático" está asignada, una de las siguientes constantes que se encuentran en el tema "**Estilos de fuente**":*

Constante	Tipo	Valor	Comentario
Automatic style sheet	Cadena	__automatic__	Se utiliza de forma predeterminada para todos los objetos
Automatic style sheet_additional	Cadena	__automatic_additional_text__	Soportado por los textos estáticos, campos y variables solamente. Se utiliza para texto adicional en las cajas de diálogo.
Automatic style sheet_main	Cadena	__automatic_main_text__	Soportado por los textos estáticos, campos y variables solamente. Se utiliza para texto principal en las cajas de diálogo.

Si el comando designa varios objetos, la hoja de estilo devuelta sólo tiene sentido si se asigna a todos los objetos.

OBJECT GET SUBFORM

OBJECT GET SUBFORM ({* ;} objeto ; puntTabla ; subFormDet {; subFormList})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
puntTabla	Tabla	⬅	Puntero a la tabla del formulario
subFormDet	Texto	⬅	Nombre del formulario detallado del subformulario
subFormList	Texto	⬅	Nombre del formulario listado del subformulario (formulario tabla)

Descripción

El comando **OBJECT GET SUBFORM** obtiene los nombres del o de los formulario(s) asociado(s) al objeto subformulario designado por los parámetros objeto y *.

Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.

En el parámetro puntTabla, el comando devuelve un puntero a la tabla del o de los formulario(s) utilizado(s). Si el subformulario utiliza un formulario proyecto, el parámetro contiene **Is nil pointer**.

En el parámetro subFormDet y(opcionalmente) en los parámetros subFormList, el comando devuelve:

- el nombre del formulario si el subformulario fue creado en el editor de formularios 4D.
- el atributo "nombre" del subformulario si el subformulario se creó a partir de un archivo .json o un objeto 4D.
  En ambos casos, si el atributo "nombre" no está definido, el comando devolverá:
  - para un archivo .json, el nombre del archivo .json (sin extensión)
  - para un objeto, "sin título"

Si no hay formulario listado, se devuelve una cadena vacía en el parámetro subFormList.

OBJECT GET SUBFORM CONTAINER SIZE

OBJECT GET SUBFORM CONTAINER SIZE (ancho ; alto)

Parámetro	Tipo		Descripción
ancho	Entero largo	⇐	Ancho del objeto subformulario
alto	Entero largo	⇐	Alto del objeto subformulario

Descripción

*El comando **OBJECT GET SUBFORM CONTAINER SIZE** devuelve el ancho y el alto (en píxeles) de un objeto subformulario "actual", mostrado en el formulario padre.*

Este comando debe llamarse desde el método de un formulario utilizado como subformulario y mostrado en un objeto subformulario. Devuelve el ancho y el alto del objeto que contiene el subformulario. Devuelve el ancho y el alto del objeto que contiene el subformulario.

Este comando es útil, por ejemplo, en el caso de objetos de subformulario que deben redimensionarse en función de las características del objeto subformulario mismo. En el evento formulario On Load, el subformulario puede llamar a este comando para calcular el espacio a su disposición con el fin de mostrar su contenido.

***Nota**: no es posible utilizar el evento On Resize directamente en el método subformulario. Ya que este evento está relacionado con el cambio de tamaño de una ventana, se genera sólo en el método del formulario padre. Puede, sin embargo, llamar explícitamente al subformulario de este evento del formulario padre utilizando, por ejemplo, el comando **EXECUTE METHOD IN SUBFORM**.*

- *Si el comando se llama desde un formulario que no se está utilizando como un subformulario, devuelve el tamaño actual de la ventana del formulario.*
- *Si el comando se llama fuera del contexto de la visualización de la pantalla (por ejemplo, durante la impresión del formulario), devuelve 0 en ancho y alto.*

OBJECT Get text orientation

OBJECT Get text orientation ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
Resultado	Entero largo	⮌	Ángulo de rotación de texto

Descripción

El comando ***OBJECT Get text orientation*** *devuelve el valor de orientación actual aplicado al texto del objeto o de los objetos designado(s) por los parámetros objeto y *.*

Puede configurar la opción "Orientación" para un objeto en modo de diseño utilizando la lista de propiedades, o utilizando el comando ***OBJECT SET TEXT ORIENTATION****.*

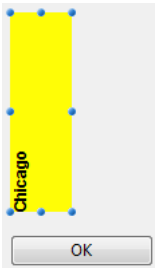
*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*El valor que se devuelve corresponde a una de las siguientes constantes, que se encuentran en el tema "****Propiedades de los objetos****":*

Constante	Tipo	Valor	Comentario
Orientation 0°	Entero largo	0	Sin rotación (valor por defecto)
Orientation 180°	Entero largo	180	Orientación del texto a 180° en el sentido horario
Orientation 90° left	Entero largo	270	Orientación del texto a 90° en el sentido antihorario
Orientation 90° right	Entero largo	90	Orientación del texto a 90° en el sentido horario

Ejemplo

Dado el siguiente objeto (donde se aplicó una orientación "90 ° a la izquierda" en el editor de formularios):



Cuando se ejecuta el formulario, si se llama a la siguiente declaración:

```
OBJECT SET TEXT ORIENTATION(*;"myText";Orientation 180°)
```

... a continuación, el objeto aparece de la siguiente manera:


```
$vOrt:=OBJECT Get text orientation(*;"myText") //$vOrt=180
```

OBJECT Get three states checkbox

OBJECT Get three states checkbox ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
Resultado	Booleano	⮌	True = casilla de selección de tres estados, False = casilla de selección estándar

Descripción

El comando ***OBJECT Get three states checkbox*** *devuelve el estado actual de la propiedad "Tres estados" de la(s) casilla(s) de selección designada(s) por los parámetros objeto y * .*

La propiedad "Tres estados" se puede definir ya sea usando la lista de propiedades, o utilizando el comando ***OBJECT SET THREE STATES CHECKBOX*** *si se llama en el proceso actual.*

OBJECT Get title

OBJECT Get title ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o campo o variable texto (si se omite *)
Resultado	Texto	⮌	Título del botón

Descripción

El comando OBJECT Get title devuelve el título (etiqueta) del objeto de formulario designado por objeto .

***OBJECT Get title** puede utilizarse con todo tipo de objetos simples que contengan una etiqueta:*

- *botones,*
- *casillas de selección*
- *botones de radio*
- *textos estáticos*
- *áreas de grupo.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

OBJECT Get type

OBJECT Get type ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Entero largo	⮌	Tipo de objeto

Descripción

El comando ***OBJECT Get type*** *devuelve el tipo del objeto designado por los parámetros objeto y * en el formulario actual .*
*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Esta sintaxis es obligatoria si está procesando objetos estáticos tales como líneas o rectángulos.*
Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena.

Nota*: si aplica este comando a un conjunto de objetos, se devuelve el tipo del último objeto.*

*El valor devuelto corresponde a una de las siguientes constantes, disponibles en el tema "****Tipos objetos formulario****:*

Constante	Tipo	Valor
Object type 3D button	Entero largo	16
Object type 3D checkbox	Entero largo	26
Object type 3D radio button	Entero largo	23
Object type button grid	Entero largo	20
Object type checkbox	Entero largo	25
Object type combobox	Entero largo	11
Object type dial	Entero largo	28
Object type group	Entero largo	21
Object type groupbox	Entero largo	30
Object type hierarchical list	Entero largo	6
Object type hierarchical popup menu	Entero largo	13
Object type highlight button	Entero largo	17
Object type invisible button	Entero largo	18
Object type line	Entero largo	32
Object type listbox	Entero largo	7
Object type listbox column	Entero largo	9
Object type listbox footer	Entero largo	10
Object type listbox header	Entero largo	8
Object type matrix	Entero largo	35
Object type oval	Entero largo	34
Object type picture button	Entero largo	19
Object type picture input	Entero largo	4
Object type picture popup menu	Entero largo	14
Object type picture radio button	Entero largo	24
Object type plugin area	Entero largo	38
Object type popup dropdown list	Entero largo	12
Object type progress indicator	Entero largo	27
Object type push button	Entero largo	15
Object type radio button	Entero largo	22
Object type radio button field	Entero largo	5
Object type rectangle	Entero largo	31
Object type rounded rectangle	Entero largo	33
Object type ruler	Entero largo	29
Object type splitter	Entero largo	36
Object type static picture	Entero largo	2
Object type static text	Entero largo	1
Object type subform	Entero largo	39
Object type tab control	Entero largo	37
Object type text input	Entero largo	3
Object type unknown	Entero largo	0
Object type view pro area	Entero largo	42
Object type web area	Entero largo	40
Object type write pro area	Entero largo	41

Ejemplo

Usted quiere cargar un formulario y obtener la lista de todos los objetos de los list boxes que contiene.

```
FORM LOAD("MyForm")
ARRAY TEXT(arrObjects;0)
FORM GET OBJECTS(arrObjects)
ARRAY LONGINT(ar_type;Size of array(arrObjects))
For($i;1;Size of array(arrObjects))
   ar_type{$i}:=OBJECT Get type(*;arrObjects{$i})
   If(ar_type{$i}=Object type listbox)
      ARRAY TEXT(arrLBObjects;0)
      LISTBOX GET OBJECTS(*;arrObjects{$i};arrLBObjects)
   End if
End for
FORM UNLOAD
```

OBJECT Get vertical alignment

OBJECT Get vertical alignment ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
Resultado	Entero largo	⮌	Tipo de alineación

Descripción

El comando ***OBJECT Get vertical alignment*** *devuelve un valor indicando el tipo de alineación vertical aplicada al objeto designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

Nota*: si aplica este comando a un conjunto de objetos, sólo se devuelve el valor de alineación del último objeto.*

El valor devuelto corresponde a una de las siguientes constantes, del tema ***Propiedades de los objetos****:*

Constante	Tipo	Valor
Align bottom	Entero largo	4
Align center	Entero largo	3
Align top	Entero largo	2

La alineación vertical puede aplicarse a los siguientes tipos de objetos de formulario:

- *list boxes,*
- *columnas de list box,*
- *encabezados y pies de list box.*

OBJECT Get visible

OBJECT Get visible ({* ;} objeto) -> Resultado

Parámetro	*Tipo*		*Descripción*
*	*Operador*	➡	*Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo*
objeto	*Objeto de formulario*	➡	*Nombre del objeto (si se especifica *) o Variable o campo (si se omite *)*
Resultado	*Booleano*	⮌	*True = objeto(s) visible(s), de lo contrario False*

Descripción

*El comando **OBJECT Get visible** devuelve True si el objeto o grupo de objetos designado por objeto tiene el atributo visible y de lo contrario False.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

OBJECT Is styled text

OBJECT Is styled text ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
Resultado	Booleano	⮌	True si el objeto es un texto multi estilo, False en caso contrario

Descripción

El comando ***OBJECT Is styled text*** *devuelve* ***True*** *sila opción "Multiestilo" está seleccionada para el(los) objeto(s) designado(s) por los parámetros objeto y * .*

La opción "Multiestilo" permite utilizar áreas de texto enriquecido incluyendo variaciones de estilo individuales. Para obtener más información, consulte la sección ***Multiestilo (área de texto enriquecido)*** *en el manual de Diseño.*

*Los objetos multiestilo se pueden administrar por programación utilizando los comandos del tema "****Texto multiestilo****":*

*Al pasar el parámetro opcional * se indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

Nota: *el comando* ***OBJECT Is styled text*** *devuelve* ***True*** *cuando se aplica a un área 4D Write Pro.*

Ejemplo

Un formulario contiene un campo representado por dos objetos diferentes, uno de los objetos tiene la propiedad "Multi-estilo" marcada, y el otro no. Usted puede escribir:

```
$Style:=OBJECT Is styled text(*;"Styled_text")
 // devuelve True ( si la opción "Multi-estilo" está seleccionada)

$Style:=OBJECT Is styled text(*;"Plain_text")
 // devuelve False (si la opción "Multi-estilo" no está seleccionada)
```

OBJECT MOVE

OBJECT MOVE ({* ;} objeto ; moveH ; moveV {; redimH {; redimV {; *}}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Campo o variable (si se omite *)
moveH	Entero largo	➡	Valor del movimiento horizontal del objeto (>0 = a la derecha, <0 = a la izquierda)
moveV	Entero largo	➡	Valor del movimiento vertical del objeto (>0 = hacia abajo, <0 = hacia arriba)
redimH	Entero largo	➡	Valor del redimensionamiento horizontal del objeto
redimV	Entero largo	➡	Valor del redimensionamiento vertical del objeto
*	Operador	➡	Si se especifica = coordenadas absolutas Si se omite = coordenadas relativas

Descripción

*El comando **OBJECT MOVE** permite mover los objetos en el formulario actual, definido por los parámetros * y objeto, moveH píxeles horizontalmente y moveV píxeles verticalmente.*

También es posible (opcionalmente) redimensionar los objetos redimH píxeles horizontalmente y redimV píxeles verticalmente.
La dirección de movimiento y redimensionamiento depende de los valores pasados en los parámetros moveH y moveV:

- *Si el valor es positivo, los objetos se mueven y redimensionan a la derecha o hacia abajo, respectivamente.*
- *Si el valor es negativo, los objetos se mueven y redimensionan a la izquierda y hacia arriba, respectivamente.*

*Si pasa el primer parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena de caracteres). Si no pasa el parámetro *, objeto es un campo o una variable. En este caso, usted no pasa una cadena sino una referencia de un campo o de una variable (campo o variable de tipo objeto únicamente).*
Si pasa un nombre de objeto en el parámetro objeto y utiliza el carácter arroba ("@") para seleccionar más de un objeto, todos los objetos seleccionados se moverán o redimensionarán.
***Nota:** desde la versión 6.5, es posible definir el modo de interpretación del carácter arroba ("@"), cuando se incluye en una cadena de caracteres. Esta opción tiene un impacto en los comandos "Propiedades de los objetos". Por favor consulte el Manual de Diseño.*
*Por defecto, los valores moveH, moveV, redimH y redimV modifican las coordenadas del objeto relativamente a su posición anterior. Si quiere que los parámetros definan las coordenadas absolutas, pase el último parámetro opcional *.*
Este comando funciona en los siguientes contextos:

- *Formularios de entrada en modo entrada de datos,*
- *Formularios mostrados utilizando el comando **DIALOG**,*
- *Encabezados y pies de página de formularios de salida mostrados por los comandos **MODIFY SELECTION** o **DISPLAY SELECTION**,*
- *Formularios en curso de impresión.*

Ejemplo 1

La siguiente instrucción mueve el botón "boton_1" 10 píxeles a la derecha, 20 píxeles hacia arriba y agranda el botón 30 píxeles de largo y 40 de alto:

```
OBJECT MOVE(*;"boton_1";10;-20;30;40)
```

Ejemplo 2

La siguiente instrucción mueve el botón "boton_1" a las siguientes coordenadas (10;20) (30;40):

```
OBJECT MOVE(*;"boton_1";10;20;30;40;*)
```

OBJECT SET ACTION

OBJECT SET ACTION ({sup ;} objeto ; accion)

Parámetro	Tipo		Descripción
sup	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
accion	Texto	➡	Acción para asociar

Descripción

*El comando **OBJECT SET ACTION** modifica, para el proceso actual, la acción estándar asociada al objeto o a los objetos designado(s) por los parámetros objeto y * .*

***Nota:** las acciones estándar también se pueden configurar para la sesión en el editor de formularios usando la lista de propiedades (ver **Acciones estándar** en el manual de Diseño).*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*En el parámetro acción, pase un cadena con el nombre de la acción estándar a asociar al objeto. Opcionalmente, la acción puede tener parámetros. Para mayor información sobre nombres de acciones, consulte la sección [#title id="964"/] en el manual de Diseño 4D. También puede pasar una de las siguientes constantes, que se encuentran en el tema "**Acción estándar**":*

[#table_kst id="3229250,3229256,3229259,3229262,3229265,3233324,3229268,3229271,3229274,3229277,3229280,3232805,3229283,322 typeCol="false"/]

***Nota de compatibilidad**: las constantes heredadas (prefijadas por _o_ en el tema) están obsoletas a partir de 4D v16 R3. Sin embargo, todavía son soportadas por compatibilidad.*

Ejemplo

*Usted desea asociar la acción estándar **Validate** con un botón:*

```
OBJECT SET ACTION(*;"bValidate";ak accept)
```

OBJECT SET AUTO SPELLCHECK

OBJECT SET AUTO SPELLCHECK ({* ;} objeto ; correcionAuto)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
correcionAuto	Booleano	➡	True = corrección automática, False= no corrección automática

Descripción

*El comando **OBJECT SET AUTO SPELLCHECK** permite definir o modificar dinámicamente el estado de la opción **Corrección ortográfica** de los objetos designados por los parámetros objeto y * para el proceso actual. Esta opción activa o desactiva la corrección ortográfica automática durante la entrada para el objeto (objetos de tipo texto únicamente).*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable o un campo. En este caso, pase una referencia en lugar de un nombre.*

*Pase **True** en correccionAuto para activar esta función para objeto y **False** para desactivarla.*

OBJECT SET BORDER STYLE

OBJECT SET BORDER STYLE ({* ;} objeto ; estiloBorde)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
estiloBorde	Entero largo	➡	Estilo de línea de borde

Descripción

El comando ***OBJECT SET BORDER STYLE*** *modifica el estilo de línea del borde del objeto(s) designada por los parámetros objeto y * .*

La propiedad "Border Line Style" modifica la apariencia de los contornos de objetos. Para obtener más información, consulte ***Estilo del borde*** *en el manual de Diseño.*

*Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo u variable de objeto únicamente).*

*En el parámetro estiloBorde, pase el valor del estilo de la línea fronteriza que desea aplicar al objeto. Puede pasar una de las siguientes constantes, que se encuentra en el tema "****Propiedades de los objetos****":*

Constante	Tipo	Valor	Comentario
Border Dotted	Entero largo	2	Los objetos aparecen enmarcados con una línea punteada de 1-pt.
Border Double	Entero largo	5	Los objetos aparecen enmarcados con una línea doble, es decir, dos líneas continuas de 1-pt. separadas por un píxel
Border None	Entero largo	0	Los objetos aparecen sin borde
Border Plain	Entero largo	1	Los objetos aparecen enmarcado con una línea de borde continua de 1-pt.
Border Raised	Entero largo	3	Los objetos aparecen con un efecto 3D (relieve)
Border Sunken	Entero largo	4	Los objetos aparecen enmarcados con un efecto 3D hundido (relieve inverso)
Border System	Entero largo	6	La línea del borde se dibuja en función de las especificaciones gráficas del sistema

OBJECT SET COLOR

OBJECT SET COLOR ({* ;} objeto ; color {; altColor})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Campo, Variable	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
color	Entero largo	➡	Nuevos colores para el objeto
altColor	Entero largo	➡	Colores alternos para un list box

Descripción

El comando **OBJECT SET COLOR** define los colores del primer plano y del fondo de los objetos de formulario especificados por objeto. Si objeto es un list box, se utiliza un parámetro adicional para definir los colores del primer plano y del fondo de las líneas pares (colores alternos).

Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa el parámetro opcional *, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de un campo o de una variable (campo o variable tipo objeto únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.

El parámetro color (como también altColor) define los colores de primer plano y fondo. El color se calcula de esta manera:

Color:=-(Primer plano+(256 * Fondo))

donde **Primer plano** y **Fondo** son números de colores (de 0 a 255) en la paleta de colores.
Color siempre es un número negativo. Por ejemplo, si el color del primer plano es 20 y el color de fondo es 10, entonces color es – (20 + (256 * 10)) o –2580.

altColor se utiliza para especificar un color alternativo para las líneas pares de un list box o de una columna de list box. En altColor, debe pasar sólo la parte del "fondo" de la fórmula de color, es decir **AltColor:=-(256 * Background)**.
Cuando este parámetro se pasa, el parámetro color se aplica sólo a las líneas impares. La utilización de colores alternos hace que las listas sean más fáciles de leer. Si objeto especifica el objeto list box, los colores alternos se utilizan en la totalidad del list box. Si objeto especifica una columna, sólo la columna utilizará los colores definidos.

Nota: puede ver la paleta de colores en la ventana Lista de propiedades del editor de formularios.

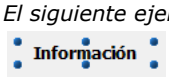
Los números de los colores más utilizados son suministrados por las siguientes constantes predefinidas, ubicadas en el tema "**Colores**":

Constante	Tipo	Valor
Black	Entero largo	15
Blue	Entero largo	6
Brown	Entero largo	13
Dark blue	Entero largo	5
Dark brown	Entero largo	10
Dark green	Entero largo	9
Dark grey	Entero largo	11
Green	Entero largo	8
Grey	Entero largo	14
Light blue	Entero largo	7
Light grey	Entero largo	12
Orange	Entero largo	2
Purple	Entero largo	4
Red	Entero largo	3
White	Entero largo	0
Yellow	Entero largo	1

Nota: mientras que **OBJECT SET COLOR** trabaja con colores indexados en la paleta de colores de 4D, el comando **OBJECT SET RGB COLORS** que le permite trabajar con cualquier color RGB. Para restablecer los colores automáticos para un objeto, utilice el comando **OBJECT SET RGB COLORS** con las constantes Default foreground color y Default background color.

Ejemplo 1

El siguiente ejemplo define el color del área de texto mostrado debajo en el editor de formularios:



Después de la ejecución de la siguiente instrucción:

```
OBJECT SET COLOR(*;"Mitexto";-(Yellow+(256*Red)))
```

... el área aparece como se ve a continuación:

Ejemplo 2

Usted quiere definir un color de fondo alterno para una columna en el list box. Puede escribir:

```
OBJECT SET COLOR(*;"countryCol";-(Dark blue+(256*Red));-(256*Orange))
```

OBJECT SET CONTEXT MENU

OBJECT SET CONTEXT MENU ({* ;} objeto ; menuContext)

Parámetro	*Tipo*		*Descripción*
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
menuContext	Booleano	➡	True = activar el menú contextual, False = desactivar el menú contextual

Descripción

*El comando **OBJECT SET CONTEXT MENU** activa o desactiva, para el proceso actual, la asociación de un menú contextual por defecto al objeto o a los objetos designado(s) por los parámetros objeto y * .*

La opción "Menú contextual" está disponible para las áreas de texto de tipo de entrada, las áreas web y las imágenes. Se puede utilizar para asociar un menú de acción estándar en función del tipo de objeto (por ejemplo, copiar/pegar para los objetos texto). Para obtener más información, consulte el manual de Diseño.

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*Pase **True** en el parámetro menuContext para activar el menú contextual, y **False** para desactivarlo.*

OBJECT SET COORDINATES

OBJECT SET COORDINATES ({* ;} objeto ; izquierda ; sup {; derecha ; inf})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Entero largo	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
izquierda	Entero largo	➡	Coordenada izquierda del objeto en píxeles
sup	Entero largo	➡	Coordenada superior del objeto en píxeles
derecha	Entero largo	➡	Coordenada derecha del objeto en píxeles
inf	Entero largo	➡	Coordenada iinferior del objeto en píxeles

Descripción

*El comando **OBJECT SET COORDINATES** modifica la ubicación y, opcionalmente, el tamaño del objeto o de los objetos designados por los parámetros objeto y * para el proceso actual.*

***Nota:** este comando es equivalente a utilizar el comando **OBJECT MOVE** y pasar el segundo parámetro *.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

En los parámetros izquierda y sup, pase las nuevas coordenadas absolutas del objeto en el formulario. Estas coordenadas deben expresarse en píxeles con respecto a la esquina superior izquierda del formulario.

También puede pasar los valores de coordenadas absolutas en los parámetros derecha e inf, que indican la esquina inferior derecha del objeto. Si esta esquina no corresponde a la esquina del objeto después de la aplicación de los parámetros izquierda y sup, el objeto cambia de tamaño en consecuencia.

***Nota:** si desea mover un objeto con respecto a su posición inicial, se recomienda utilizar el comando existente **OBJECT MOVE**.*

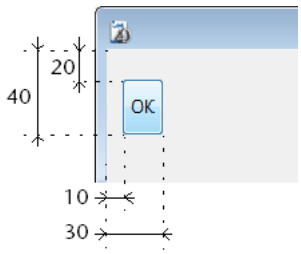
Este comando sólo funciona en los siguientes contextos:

- *Los formularios de entrada en el modo entrada,*
- *Formulario mostrados usando el comando **DIALOG**,*
- *Encabezados y pies de página de formularios de salida mostrados por el comando **MODIFY SELECTION** o **DISPLAY SELECTION**,*
- *Los formularios en impresión.*

Ejemplo

La siguiente declaración ubica el objeto "button_1" en las coordenadas (10,20) (30,40):

```
OBJECT SET COORDINATES(*;"button_1";10;20;30;40)
```



OBJECT SET CORNER RADIUS

OBJECT SET CORNER RADIUS ({* ;} objeto ; radio)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
radio	Entero largo	➡	Nuevo radio de esquinas redondeadas (en píxeles)

Descripción

*El comando **OBJECT SET CORNER RADIUS** modifica el radio de las esquinas de los objetos rectángulo redondeado cuyos nombres pasó en el parámetro objeto. El nuevo radio sólo se define para el proceso y no se guarda en el formulario.*

*Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

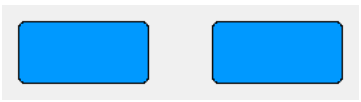
***Nota:** en la versión actual de 4D, este comando sólo se aplica a los rectángulos redondeados (que son objetos estáticos). Como resultado, sólo la sintaxis basada en el nombre del objeto (usando el parámetro *) es compatible.*

En el parámetro radio, pase un nuevo valor del radio en píxeles a aplicar en las esquinas del objeto. Por defecto, este valor es de 5 píxeles.

***Nota:** también puede modificar este valor a nivel de formulario utilizando la lista de propiedades (ver **Nueva propiedad Redondeado de esquinas para rectángulo redondeado**).*

Ejemplo

Usted tiene los siguientes rectángulos en su formulario, llamados respectivamente "Rect1" y "Rect2":



Puede ejecutar el siguiente código para cambiar su radio:

```
OBJECT SET CORNER RADIUS(*;"Rect@";20)
```

OBJECT SET DATA SOURCE

OBJECT SET DATA SOURCE ({* ;} objeto ; fuenteDatos)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
fuenteDatos	Puntero	➡	Puntero a la nueva fuente de datos del objeto

Descripción

El comando ***OBJECT SET DATA SOURCE*** *modifica la fuente de datos de los objetos designados por los parámetros objeto y * .*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

La fuente de datos es el campo o la variable cuyo valor es representado por el objeto cuando se ejecuta el formulario. En modo Diseño, la fuente de datos se define en la lista de propiedades, por lo general a través de las líneas Fuente y Campo fuente (campos) o Nombre de variable (variables):

Objetos	
Tipo	Campo
Nombre del objeto	Campo5
Fuente de datos	
Tabla fuente	EMPLEADOS
Campo fuente	Apellido

Objetos	
Tipo	Botón
Nombre del objeto	Botón1
Nombre de la variable	vB1
Título	Colores

A excepción de los list box (ver más adelante), todas las fuentes de datos del formulario pueden ser modificadas por este comando. Es responsabilidad del desarrollador asegurar la consistencia de los cambios realizados.

En el caso de los list box, se deben tener en cuenta los siguientes puntos:

- *Los cambios de fuentes de datos deben tener en cuenta el tipo de list box: por ejemplo, no es posible utilizar un campo como fuente de datos de una columna de en un list box de tipo array.*
- *Para los list box de tipo selección, no es posible modificar o leer la fuente de datos del objeto list box en sí: en este caso, se trata de una referencia interna y no de una fuente de datos.*
- *Este comando se utiliza en el contexto de los list box de tipo array. Para los list box de tipo selección, puede en vez utilizar el comando [**LISTBOX SET COLUMN FORMULA***

Si este comando se aplica a una fuente de datos que no se puede editar, no hace nada.

Ejemplo

Cambio de la fuente de datos para un área de entrada:

```
C_POINTER($ptrField)
$ptrField:=Field(3;2)
OBJECT SET DATA SOURCE(*;"Input";$ptrField)
```

OBJECT SET DRAG AND DROP OPTIONS

OBJECT SET DRAG AND DROP OPTIONS ({* ;} objeto ; arrastrable ; arrastrableAuto ; soltable ; soltableAuto)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
arrastrable	Booleano	➡	0 = False, 1 = True
arrastrableAuto	Booleano	➡	0 = False, 1 = True
soltable	Booleano	➡	0 = False, 1 = True
soltableAuto	Booleano	➡	0 = False, 1 = True

Descripción

*El comando **OBJECT SET DRAG AND DROP OPTIONS** define o modifica dinámicamente las opciones de arrastrar y soltar para el objeto o los objetos designados por los parámetros objeto y * para el proceso actual.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, pase una referencia de variable en lugar de una cadena.*

En cada parámetro, pase un booleano indicando si la opción correspondiente está activa o inactiva:

- *arrastrable = True: objeto arrastrable en modo programado.*
- *arrastrableAuto = True (utilizable únicamente con los campos y variables texto, combo boxes y list boxes): objeto arrastrable en modo automático.*
- *soltable = True: objeto acepta soltar en modo programado.*
- *soltableAuto = True (utilizable únicamente con los campos y variables imagen, texto, combo boxes y list boxes): objeto acepta soltar en modo automático.*

Ejemplo

Definición de un área de texto en arrastrar y soltar auto:

```
OBJECT SET DRAG AND DROP OPTIONS(*;"Comments";False;True;False;True)
```

OBJECT SET ENABLED

OBJECT SET ENABLED ({* ;} objeto ; activo)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable (se se omite *)
activo	Booleano	➡	True = objeto(s) activo(s), de lo contrario False

Descripción

*El comando **OBJECT SET ENABLED** utilizado para activar o desactivar el objeto o grupo de objetos especificado por objeto en el formulario actual.*

Un objeto activo reacciona a los clics y atajos de teclado.

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable (variable objeto únicamente) en lugar de una cadena.*

Pase True en el parámetro activo para activar los objetos y False para desactivarlas.

Este comando se puede aplicar a los siguientes tipos de objetos:

- *Botón, Botón por defecto, Botón 3D, Botón invisible, Botón inverso*
- *Radio botón, Botón de radio 3D, Botón Imagen*
- *Casilla de selección, Casilla de selección 3D*
- *Menú pop-up , Lista desplegable, Combo box, Menú/lista desplegable*
- *Termómetro, Regla*

***Nota**: este comando no tiene efecto con un objeto al que se le ha asignado una acción estándar (4D se encarga de modificar el estado de este objeto cuando sea necesario), excepto en el caso de las acciones Validar y Cancelar.*

OBJECT SET ENTERABLE

OBJECT SET ENTERABLE ({* ;} objeto ; editable)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Tabla o campo o variable (si se omite *)
editable	Booleano	➡	True para editable; False para no editable

Descripción

*El comando **OBJECT SET ENTERABLE**]vuelve editables o no editables los objetos de formulario especificados por objeto.*

*Si especifica el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si omite el parámetro opcional *, indica que el parámetro objeto es una tabla, un campo o una variable. En este caso, usted especifica una referencia de un campo o de una variable (campo o variable tipo objeto únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.*

La utilización de este comando es equivalente a la selección de la opción editable para un campo o una variable en la lista de propiedades del editor de formularios. Este comando funciona en subformularios únicamente si se encuentra en el método formulario del subformulario.

Cuando areaEntrada es editable (TRUE), el usuario puede mover el cursor en el área e introducir datos. Cuando areaEntrada es no editable (FALSE), el usuario no puede mover el cursor en el área y no puede introducir datos.

*El comando **OBJECT SET ENTERABLE***
*también puede utilizarse para activar por programación el modo "Editable en lista" para los subformularios y formularios listados mostrados utilizando los comandos **MODIFY SELECTION** y **DISPLAY SELECTION**:*

- *Para los subformularios, en el parámetro areaEntrada, pase el nombre de la tabla del subformulario o el nombre del objeto del subformulario, por ejemplo: **OBJECT SET ENTERABLE**(*;"Subform";True).*
- *Para los formularios listados, debe pasar el nombre de la tabla del formulario en el parámetro areaEntrada, por ejemplo: **OBJECT SET ENTERABLE**([MiTabla];True).*

Volver un objeto no editable no evita que cambie su valor por programación.

***Nota:** para volver una celda de list box no editable, pase el valor -1 a $0 en el evento On Before Data Entry, ver **Gestión de entrada**.*

Ejemplo 1

El siguiente ejemplo define un campo de envío, dependiendo del peso del paquete. Si el paquete pesa un 1 kilo o menos, el envío se realiza a través de la Oficina Postal Nacional y el campo no es editable. De lo contrario, el campo es editable.

```
If([Envio]Peso<=1)
   [Envio]Empresa:="Oficina Postal Nacional"
   OBJECT SET ENTERABLE([Envio]Empresa;False)
Else
   OBJECT SET ENTERABLE([Envio]Empresa;True)
End if
```

Ejemplo 2

Este es el método de objeto de una casilla de selección ubicada en el encabezado de una lista para controlar el modo Entrada en lista:

```
C_BOOLEAN(bEditable)
OBJECT SET ENTERABLE([Tabla1];bEditable)
```

OBJECT SET EVENTS

OBJECT SET EVENTS ({* ;} objeto ; arrEventos ; modo)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto o "" para designar el formulario (si * se especifica) o Campo o variable (si * se omite)
arrEventos	Array entero largo	➡	Array de eventos a definir
modo	Entero largo	➡	Modo de activación de los eventos definidos en arrEvents

Descripción

El comando ***OBJECT SET EVENTS*** *modifica, para el proceso actual, la configuración de los eventos formulario del formulario, de los objetos designados por los parámetros objeto y * .*

*Si pasa el parámetro opcional * se indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*
*Para definir la configuración de eventos para el formulario, pase el parámetro opcional * y una cadena vacía "" en objeto, en este caso, usted designa el formulario actual.*

Nota: *si desea modificar los eventos de un subformulario relacionado con una tabla, sólo puede utilizar la sintaxis basada en el nombre del objeto.*

*En el parámetro arrEvents, pase un array entero largo que contenga la lista de eventos de formulario predefinidos o personalizados que desea modificar (se puede utilizar el parámetro modo para especificar si la modificación consiste en la activación o desactivación de los eventos) . Para designar un evento predefinido a modificar, puede pasar en cada elemento del array arrEvents, una de las siguientes constantes, que se encuentra en el tema "**Eventos de formulario**":*

Constante	Tipo	Valor	Comentario
On Activate	Entero largo	11	La ventana del formulario se convierte en la ventana del primer plano
On After Edit	Entero largo	45	El contenido del objeto editable que tiene el foco acaba de ser modificado
On After Keystroke	Entero largo	28	Un carácter está apunto de introducirse en el objeto que tiene el foco. **Get edited text** devuelve el contenido incluyendo este carácter
On After Sort	Entero largo	30	(List box únicamente) Se acaba de efectuar una ordenación estándar en una columna del list box
On Arrow Click	Entero largo	38	(Botones 3D únicamente) El área "flecha" de un botón 3D recibe un clic
On Before Data Entry	Entero largo	41	(List box únicamente) Una celda de list box está a punto de pasar a modo edición
On Before Keystroke	Entero largo	17	Un carácter está a punto de introducirse en el objeto que tiene el foco. **Get edited text** devuelve el texto del objeto sin este carácter.
On Begin Drag Over	Entero largo	46	Se va a arrastrar un objeto
On Begin URL Loading	Entero largo	47	(Áreas web únicamente) Un nuevo URL se carga en el área web
On bound variable change	Entero largo	54	Se modifica la variable relacionada a un subformulario.
On Clicked	Entero largo	4	Ocurre un clic sobre un objeto
On Close Box	Entero largo	22	Se ha hecho clic en la casilla de cerrar la ventana
On Close Detail	Entero largo	26	Se cierra el formulario de entrada y regresa al formulario de salida
On Collapse	Entero largo	44	(Listas jerárquicas únicamente) Un elemento de la lista jerárquica se ha contraído vía un clic o una tecla
On Column Moved	Entero largo	32	(List box únicamente) El usuario mueve una columna de list box vía arrastrar y soltar
On Column Resize	Entero largo	33	(List box únicamente) El ancho de una columna de list box es modificado por un usuario con el ratón
On Data Change	Entero largo	20	Se han modificado los datos de un objeto
On Deactivate	Entero largo	12	La ventana del formulario deja de ser la ventana del primer plano
On Delete Action	Entero largo	58	(Listas jerárquicas y list box únicamente). El usuario solicita borrar un elemento.
On Display Detail	Entero largo	8	Un registro se va a mostrar en una lista o una línea se va a mostrar en un list box.
On Double Clicked	Entero largo	13	Un objeto ha recibido un doble clic
On Drag Over	Entero largo	21	Los datos pueden soltarse en un objeto
On Drop	Entero largo	16	Se han soltado datos en un objeto
On End URL Loading	Entero largo	49	(Áreas web únicamente) Se han cargado todos los recursos del URL
On Expand	Entero largo	43	(Listas jerárquicas únicamente) Se ha expandido un elemento de la lista jerárquica utilizando un clic o una tecla
On Footer Click	Entero largo	57	(List box únicamente) Un clic en el pie de un list box o de una columna de list box
On Getting Focus	Entero largo	15	Un objeto de formulario toma el foco
On Header	Entero largo	5	El encabezado del formulario se va a imprimir o a mostrar
On Header Click	Entero largo	42	(List box únicamente) Ocurre un clic en un encabezado de columna del list box
On Load Record	Entero largo	40	En modo entrada en lista, se carga un registro durante modificación (el usuario hace clic en una línea del registro y un campo pasa a modo edición)
On Long Click	Entero largo	39	(Botones 3D únicamente) Se hace clic en un botón 3D y el botón del ratón permanece presionado por un cierto tiempo
On Losing Focus	Entero largo	14	Un objeto de formulario está perdiendo el foco
On Mac toolbar button	Entero largo	55	El usuario hace clic en el botón de gestión de la barra de herramientas en Mac OS.
On Menu Selected	Entero largo	18	Se ha seleccionado un comando de menú
On Mouse Enter	Entero largo	35	El cursor del ratón entra al área gráfica de un objeto

Constante	Tipo	Valor	Comentario
On Mouse Leave	Entero largo	36	El cursor del ratón sale del área gráfica de un objeto
On Mouse Move	Entero largo	37	El cursor del ratón se mueve al menos un píxel O cuando se presiona una tecla de modificación (Ctrl, Alt, Bloq mayús). Si el evento está seleccionado para un objeto únicamente, se genera sólo cuando el cursor se encuentra dentro del área gráfica del objeto
On Open Detail	Entero largo	25	El formulario detallado asociado con el formulario de salida o con el listbox está apunto de ser abierto
On Open External Link	Entero largo	52	(Áreas web únicamente) Se ha abierto un URL externo en el navegador
On Outside Call	Entero largo	10	El formulario recibe una llamada **POST OUTSIDE CALL**
On Picture Scroll	Entero largo	59	El usuario desplaza el contenido de un campo o de una variable imagen utilzando el ratón o una tecla.
On Plug in Area	Entero largo	19	Un objeto externo solicitó que se ejecute su método de objeto
On Printing Break	Entero largo	6	Se va a imprimir una de las áreas de ruptura del formulario
On Printing Detail	Entero largo	23	Se va a imprimir el área de detalle del formulario
On Printing Footer	Entero largo	7	Se va a imprimir el área de pie de página del formulario
On Resize	Entero largo	29	La ventana del formulario se redimensiona
On Row Moved	Entero largo	34	(List box únicamente) El usuario mueve una fila de un list box vía arrastrar y soltar
On Selection Change	Entero largo	31	• List box: se modifica la selección actual de líneas o columnas • Registros en lista: se modifica el registro actual o la selección actual de líneas en un formulario listado o en un subformulario • Lista jerárquica: la selección en la lista se modifica luego de un clic o de presionar una tecla • Variable o campo editable: la selección de texto o la posición del cursor en el área se modifica al hacer clic o presionar una tecla.
On Timer	Entero largo	27	El número de tics definido por el comando **SET TIMER** se ha pasado
On Unload	Entero largo	24	El formulario se cierra y libera
On URL Filtering	Entero largo	51	(Áreas web únicamente) Un URL fue bloqueado por el área web
On URL Loading Error	Entero largo	50	(Áreas web únicamente) Ocurrió un error cuando se cargaba el URL
On URL Resource Loading	Entero largo	48	(Áreas web únicamente) Se carga un nuevo recurso en el área web
On Validate	Entero largo	3	Se ha valido la entrada de datos en el registro
On Window Opening Denied	Entero largo	53	(Áreas web únicamente) Se ha bloqueado una ventana pop-up

Es importante tener en cuenta que el evento On Load no está incluido en esta lista: este evento no se puede definir porque ya se ha generado durante la ejecución del comando.

*En arrEvents, también puede pasar todo valor correspondiente a un evento personalizado. En este caso, recomendamos utilizar valores negativos (ver el comando **CALL SUBFORM CONTAINER**).*

*El parámetro modo se utiliza para definir el tratamiento global a efectuar para los elementos del array. Para ello, puede pasar una de las siguientes constantes, que se encuentra en el tema "**Propiedades de los objetos**":*

Constante	Tipo	Valor	Comentario
Disable events others unchanged	Entero largo	2	Todos los eventos listados en el array arrEvents se desactivan; el estado de todos los demás eventos no cambia
Enable events disable others	Entero largo	0	Todos los eventos listados en el array arrEvents se activan; todos los demás eventos se desactivan
Enable events others unchanged	Entero largo	1	Todos los eventos listados en el array arrEvents se activan; el estado de todos los demás eventos no cambia

*El comando **OBJECT SET EVENTS** puede dar lugar a la activación de eventos que no son compatibles con el objeto (dependiendo del tipo). En este caso, simplemente se ignoran los eventos.*

*Si un objeto se duplica después de una llamada al comando **OBJECT SET EVENTS**, la configuración resultante de activación/desactivación también se duplica.*

Ejemplo 1

Activación de tres eventos formulario para un conjunto de objetos list box y desactivación de otros eventos:

```
ARRAY LONGINT($MyEventsOnLB;3)
$MyEventsOnLB {1}:=On After Sort
$MyEventsOnLB {2}:=On Column Moved
$MyEventsOnLB {3}:=On Column Resize
OBJECT SET EVENTS(*;"MyLB@";$MyEventsOnLB;Enable events disable others)
 // activa 3 eventos y desactiva todos los demás
```

Ejemplo 2

Desactivación de tres eventos formulario para un conjunto de objetos list box, sin modificar los otros eventos:

```
ARRAY LONGINT($MyEventsOnLB;3)
$MyEventsOnLB {1}:=On After Sort
$MyEventsOnLB {2}:=On Column Moved
$MyEventsOnLB {3}:=On Column Resize
OBJECT SET EVENTS(*;"MyLB@";$MyEventsOnLB;Disable events others unchanged)
 // desactiva solo 3 eventos
```

Ejemplo 3

Activación de un evento formulario para un objeto, sin modificar los otros eventos:

```
ARRAY LONGINT($MyEventsOnLB;1)
$MyEventsOnLB {1}:=On Column Moved
OBJECT SET EVENTS(*;"Col1";$MyEventsOnLB;Enable events others unchanged)
 // activa únicamente el evento
```

Ejemplo 4

Desactivación de todos los eventos del formulario:

```
ARRAY LONGINT($MyFormEvents;0)
OBJECT SET EVENTS(*;"";$MyFormEvents;Enable events disable others)
 // desactiva todos los eventos
```

Ejemplo 5

Desactivación de un solo evento del formulario sin modificar los otros:

```
ARRAY LONGINT($MyFormEvents;1)
$MyFormEvents{1}:=On Timer
OBJECT SET EVENTS(*;"";$MyFormEvents;Disable events others unchanged)
 // solo desactiva el evento
```

OBJECT SET FILTER

OBJECT SET FILTER ({* ;} objeto ; filtroEntrada)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
filtroEntrada	Cadena	➡	Nuevo filtro de entrada para el área editable

Descripción

OBJECT SET FILTER *reemplaza el filtro de entrada para los objetos especificados por objeto por filtroEntrada.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si omite el parámetro opcional *, indica que el parámetro objeto es un campo o una variable. En este caso, usted especifica una referencia de un campo o de una variable (campo o variable de tipo objeto únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección* ***Propiedades de los objetos****.*

OBJECT SET FILTER *puede utilizarse en formularios de entrada y diálogos y puede aplicarse a los campos y variables editables que aceptan un filtro de entrada en el entorno Diseño.*

Al pasar una cadena vacía en filtroEntrada se elimina el filtro de entrada actual para los objetos.

Nota: *este comando no puede utilizarse con campos ubicados en el formulario listado de un subformulario.*

Nota: *en filtroEntrada, para utilizar filtros de entrada predefinidos utilizando la Caja de herramientas, coloque un prefijo en el filtro de entrada, una barra vertical (|).*

Ejemplo 1

El siguiente ejemplo define el filtro de entrada para el campo código postal. Si la dirección es de España, el filtro se define para los códigos postales españoles. De lo contrario, puede aceptar todo valor de entrada:

```
If([Empresas]Pais ="ES") ` Definir el filtro para un formato del código postal español
   OBJECT SET FILTER([Empresas]Codigo Postal;"&9#####")
Else ` Definir el filtro para aceptar todo valor alfanumérico y mayúsculas
   OBJECT SET FILTER([Empresas]Codigo Postal;"~@")
End if
```

Ejemplo 2

El siguiente ejemplo permite únicamente la entrada de las letras "a," "b," "c," o "g" en un campo de dos letras:

```
OBJECT SET FILTER([Tabla]Campo ;"&"+Char(Double quote)+"a;b;c;g"+Char(Double quote)+"##")
```

Nota: *este ejemplo define el filtro de entrada &"a;b;c;g"##.*

OBJECT SET FOCUS RECTANGLE INVISIBLE

OBJECT SET FOCUS RECTANGLE INVISIBLE ({* ;} objeto ; invisible)

Parámetro	*Tipo*		*Descripción*
*	*Operador*	➡	*Si se especifica, objeto es un nombre de objeto (cadena)* *Si se omite, objeto es una variable*
objeto	*Objeto de formulario*	➡	*Nombre de objeto (si se especifica *) o* *Variable (si se omite *)*
invisible	*Booleano*	➡	*True = rectángulo de foco oculto,* *False = rectángulo de foco visible*

Descripción

*El comando **OBJECT SET FOCUS RECTANGLE INVISIBLE** permite definir o modificar dinámicamente la opción invisibilidad del rectángulo de foco del objeto designado por los parámetros objeto y * para el proceso actual. Esta configuración corresponde a la opción **Ocultar rectángulo de foco** disponible para los objetos editables en la Lista de propiedades en modo Diseño.*

***Nota**: sólo puede utilizar esta opción bajo Mac OS. No tiene efecto bajo Windows.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable o un campo. En este caso, se pasa una referencia de variable en lugar de una cadena.*

*Pase **True** en el parámetro invisible para ocultar el rectángulo de foco y **False** para dejarlo visible.*

OBJECT SET FONT

OBJECT SET FONT ({* ;} objeto ; fuente)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
fuente	Cadena	➡	Nombre o número de fuente

Descripción

OBJECT SET FONT muestra objeto utilizando la fuente especificada en el parámetro fuente. El parámetro fuente debe contener un nombre de fuente válido.

Si especifica el parámetro opcional *, indica un nombre de objeto (una cadena) en objeto. Si omite el parámetro opcional *, indica que el parámetro objeto es un campo o una variable. En este caso, se especifica una referencia de campo o de variable (campo o variable objeto únicamente) en lugar de una cadena.

Ejemplo 1

El siguiente ejemplo define la fuente de un botón llamado bOK:

```
OBJECT SET FONT(bOK;"Arial")
```

Ejemplo 2

El siguiente ejemplo define la fuente para todos los objetos de formulario cuyo nombre contenga "info":

```
OBJECT SET FONT(*;"@info@";"Times")
```

Ejemplo 3

El siguiente ejemplo utiliza la opción especial %password, diseñada para la entrada y visualización de campos de tipo "contraseña". Cuando pase "%password" en el parámetro fuente:

- cada carácter introducido en el objeto se muestra con el mismo simbolo,
- las acciones "copiar" y "pegar" se desactivan en el objeto.

Nota: puede utilizar la opción %password con los objetos de tipo campo, variable y combo box.

```
OBJECT SET FONT([Users]Password;"%password")
```

OBJECT SET FONT SIZE

OBJECT SET FONT SIZE ({* ;} objeto ; tamaño)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
tamaño	Entero largo	➡	Tamaño de fuente en puntos

Descripción

OBJECT SET FONT SIZE *define el tamaño de la fuente de los objetos de formulario especificados por objeto utilizando el tamaño de fuente que se pasa en tamaño.*

*Si especifica el parámetro opcional *, indica un nombre de objeto (una cadena) en objeto. Si omite el parámetro opcional *, indica que el parámetro objeto es un campo o una variable. En este caso, usted especifica una referencia de un campo o de una variable (objetos campo o variable únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección .*

El tamaño puede ser cualquier entero entre 1 y 255. Si el tamaño de fuente exacto no existe, los caracteres son redimensionados proporcionalmente.

El área para el objeto, como se definió en el formulario, debe ser lo suficientemente grande para mostrar los datos en el nuevo tamaño. De lo contrario, el texto puede truncarse o no ser visualizado.

Ejemplo 1

El siguiente ejemplo define el tamaño de fuente para una variable llamada vtInfo:

```
OBJECT SET FONT SIZE(vtInfo;14)
```

Ejemplo 2

El siguiente ejemplo define el tamaño de fuente para todos los objetos de formulario cuyo nombre comienza por "hl":

```
OBJECT SET FONT SIZE(*;"hl@";14)
```

OBJECT SET FONT STYLE

OBJECT SET FONT STYLE ({* ;} objeto ; estilos)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
estilos	Entero largo	➡	Estilo de fuente

Descripción

OBJECT SET FONT STYLE *asigna el estilo de fuente de estilos a los objetos de formulario especificados por objeto.*

*Si especifica el parámetro opcional *, indica un nombre de objeto (una cadena) en objeto. Si omite el parámetro opcional *, indica que el parámetro objeto es un campo o una variable. En este caso, se especifica una referencia de un campo o de una variable (campo o variable tipo objeto únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección .*

En estilos se pasa una de las constantes predefinidas que define la selección de estilo de fuente. Las siguientes son constantes predefinidas de 4D:

Constante	Tipo	Valor
Plain	Entero largo	0
Bold	Entero largo	1
Italic	Entero largo	2
Underline	Entero largo	4

Ejemplo 1

Este ejemplo define el estilo de fuente para un botón llamado bAñadirNuevo. El estilo de fuente definido es negrita itálica:

```
OBJECT SET FONT STYLE(bAñadirNuevo;Bold+Italic)
```

Ejemplo 2

Este ejemplo define el estilo de fuente Plain para todos los objetos de formulario que comienzan por "vt":

```
OBJECT SET FONT STYLE(*;"vt@";Plain)
```

OBJECT SET FORMAT

OBJECT SET FORMAT ({* ;} objeto ; formato)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
formato	Cadena	➡	Nuevo formato de salida para el objeto

Descripción

OBJECT SET FORMAT reemplaza el formato de salida para los objetos especificados por objeto por el formato que pasa en formatSalida. El nuevo formato se utiliza únicamente para la visualización actual, no se guarda con el formulario.

Si especifica el parámetro opcional *, indica un nombre de objeto (una cadena) en objeto. Si omite el parámetro opcional *, indica que el parámetro objeto es un campo o una variable. En este caso, se especifica una referencia de un campo o de una variable (campo o variable tipo objeto únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.

OBJECT SET FORMAT puede utilizarse en formularios de entrada y de salida (visualizados o impresos) y pueden aplicarse a campos o variables (editables/no editables).

Naturalmente, debe utilizar un formato de salida compatible con el tipo de datos presente en el objeto o con el objeto mismo.

Booleanos

Para dar formato a campos booleanos, hay dos posibilidades:

- Puede pasar un valor simple en formatSalida. En este caso, el campo se mostrará como una casilla de selección y su etiqueta será el valor especificado.
- Puede pasar dos valores, separados por un punto y coma (;), en formatSalida. En este caso, el campo se mostrará como dos botones radio.

Fechas

Para dar formato a los campos o variables de tipo Fecha, pase **Char(n)** en formatSalida, donde n es una de las siguientes constantes predefinidas de 4D:

Constante	Tipo	Valor	Comentario
Blank if null date	Entero largo	100	"" en lugar de 0 en caso de valor nulo. Esta constante debe adicionarse al formato de visualización.
Date RFC 1123	Entero largo	10	Fri, 10 Sep 2010 13:07:20 GMT
Internal date abbreviated	Entero largo	6	29 dic, 2006
Internal date long	Entero largo	5	29 diciembre 2006
Internal date short	Entero largo	7	
Internal date short special	Entero largo	4	12/29/06 (pero 12/29/1896 o 12/29/2096)
ISO Date	Entero largo	8	2006-12-29T00:00:00 (obsoleto)
ISO Date GMT	Entero largo	9	2010-09-13T16:11:53Z
System date abbreviated	Entero largo	2	dom. 29 de 2006
System date long	Entero largo	3	domingo 29 diciembre 2006
System date short	Entero largo	1	

Nota: la constante **Blank if null** debe añadirse al formato, ella indica que en caso de un valor nulo 4D debe mostrar una zona vacía en lugar de ceros.

Horas

Para dar formato a los campos o variables de tipo Hora, pase **Char(n)** en formatSalida, donde n es una de las siguientes constantes predefinidas de 4D:

Constante	Tipo	Valor	Comentario
HH MM SS	Entero largo	1	
HH MM	Entero largo	2	
Hour min sec	Entero largo	3	1 hora 2 minutos 3 segundos
Hour min	Entero largo	4	1 hora 2 minutos
HH MM AM PM	Entero largo	5	
MM SS	Entero largo	6	
Min sec	Entero largo	7	62 minutos 3 segundos
ISO time	Entero largo	8	
System time short	Entero largo	9	
System time long abbreviated	Entero largo	10	1•02•03 AM (Mac únicamente)
System time long	Entero largo	11	1:02:03 AM HNEC (Mac únicamente)
Blank if null time	Entero largo	100	"" en lugar de 0

Nota: la constante **Blank if null** debe añadirse al formato, ella indica que en caso de un valor nulo 4D debe mostrar una zona vacía en lugar de ceros.

Imágenes

Para dar formato a campos o variables de tipo Imagen, pase **Char(n)** en formatSalida, donde n es una de las siguientes constantes predefinidas de 4D:

Constante	Tipo	Valor
Truncated centered	Entero largo	1
Scaled to fit	Entero largo	2
On background	Entero largo	3
Truncated non centered	Entero largo	4
Scaled to fit proportional	Entero largo	5
Scaled to fit prop centered	Entero largo	6
Replicated	Entero largo	7

Alfas y numéricos

Para dar formato a campos o variables de tipo alfa o numérico, pase directamente la etiqueta del formato en el parámetro formatSalida.

Para mayor información sobre formatos de salida, consulte las secciones **Formatos numéricos** y **Formatos Alfa** en el manual de Diseño de 4D.

Nota: en formatSalida, para utilizar los formatos de salida personalizados que usted haya podido crear en la caja de diálogo Preferencias, coloque un prefijo al nombre del formato con una barra vertical (|).

Botones imagen

Para dar formato a botones imagen, pase en el parámetro formatSalida una cadena de caracteres respetando la siguiente sintaxis:
cols;lineas;imagen;modo{;ticks}

- cols = número de columnas en la imagen.
- lineas = número de líneas en la imagen.
- imagen= imagen utilizada, proveniente de la librería de imágenes o de una variable imagen:
  - Si la imagen proviene de la librería de imágenes, introduzca su número, precedido de un signo de interrogación (ej.: "?250").
  - Si la imagen proviene de una variable imagen, introduzca el nombre de la variable.
- modo = modo de visualización y de funcionamiento del botón imagen. Este parámetro puede tomar cualquiera de los siguientes valores: 0, 1, 2, 16, 32, 64 y 128. Cada uno de estos valores representa un modo de visualización o de funcionamiento. Estos valores son acumulativos, por ejemplo, si quiere activar los modos 1 y 64, pase 65 en el parámetro flags. Estos son los detalles para cada valor:
  - modo = 0 (ninguna opción)
  Muestra la imagen siguiente en la serie cuando el usuario hace clic en la imagen. Muestra la imagen anterior en la serie cuando el usuario presiona la tecla Mayús y hace clic en la imagen. Cuando el usuario alcanza la última imagen en la serie, la imagen no cambia cuando el usuario hace clic nuevamente. Es decir que no va a la primera imagen de la serie.
  - modo = 1 (Cambiar continuamente)
  Similar al anterior, excepto que cuando el usuario hace clic en la imagen y mantiene el botón del ratón presionado la visualización de imágenes es continua (como una animación). Cuando el usuario alcanza la última imagen, el objeto no regresa a la primera imagen.
  - modo = 2 (Volver al inicio)
  Similar al anterior, a diferencia de que las imágenes se muestran en un bucle continuo. Cuando el usuario llega a la última imagen y hace clic nuevamente, la primera imagen aparece, y así sucesivamente.
  - modo = 16 (Cambiar al pasar el cursor encima)
  El contenido del botón imagen se modifica cuando el cursor del ratón pasa sobre él. La imagen inicial se restablece cuando el cursor deja el área del botón. Este modo se utiliza con frecuencia en aplicaciones multimedia o en documentos HTML. La imagen que se muestra es la última imagen de la tabla de miniaturas, a menos que la opción Última imagen si desactivado esté seleccionada (128). Si esa opción está seleccionada, se muestra la miniatura anterior a la última.
  - modo = 32 (Volver al soltar el clic)
  Este modo funciona con dos imágenes. Muestra la primera imagen todo el tiempo excepto cuando el usuario hace clic en el botón. En ese caso, la segunda imagen se muestra hasta que el botón del ratón es liberado. Este modo le permite crear un botón de acción que muestra su estatus (normal o presionado). Puede utilizar este modo para crear un efecto 3D o para mostrar toda imagen que simbolice la acción.
  - modo = 64 (Transparente)

Utilizado para volver transparente el fondo de la imagen.
- modo = 128 (Última imagen si desactivado)

Este modo le permite definir que la última miniatura debe ser mostrada cuando el botón esté inactivo. Cuando este modo es seleccionado, 4D muestra la última miniatura cuando el botón está desactivado. Cuando este modo se utiliza con los modos 0, 1 y 2, la última miniatura no se tiene en cuanta en la secuencia de los otros modos. Aparecerá únicamente cuando el botón esté desactivado.
• ticks = activación del modo "Cambiar cada x ticks" y define intervalo de tiempo entre la visualización de cada imagen. Cuando se pasa este parámetro opcional, le permite hacer ciclos a través del contenido del botón imagen a la velocidad especificada. Por ejemplo, si usted introduce "2;3;?16807;0;10", el botón imagen mostrará una imagen diferente cada 10 tics. Cuando este modo está activo, sólo el modo Transparente puede utilizarse (64).

Menús imagen desplegable

Para dar formato a los menús imagen desplegable, pase en el parámetro formatSalida una cadena de caracteres respetando la siguiente sintaxis:
cols;lineas;imagen;hMargen;vMargen;modo

- *cols = número de columnas de la imagen.*
- *lineas = número de líneas de la imagen.*
- *imagen = imagen utilizada, proveniente de la librería de imágenes o de una variable imagen:*
  - si la imagen proviene de la librería de imágenes, introduzca su número, precedido por un signo de interrogación (ej. : "?250").
  - Si la imagen proviene de una variable imagen, introduzca el nombre de la variable.
- *hMargen = margen en píxeles entre los límites horizontales del menú y la imagen.*
- *vMargen = margen en píxeles entre los límites verticales del menú y la imagen.*
- *modo = modo de transparencia de menú imagen desplegable. Acepta los valores 0 y 64:*
  - modo = 0: el menú imagen desplegable no es transparente,
  - modo = 64: el menú imagen desplegable es transparente.

Termómetros y reglas

Para dar formato a objetos de tipo termómetro o regla, pase en el parámetro formatSalida, una cadena de carácter respetando la siguiente sintaxis:
min;max;unidad;paso;modo{;format{;visualización}}

- *min = valor de la graduación de origen del indicador.*
- *max = valor de la graduación final del indicador.*
- *unid = intervalo entre las graduaciones del indicador.*
- *interv = intervalo mínimo del movimiento del cursor en el indicador.*
- *modo = modo de visualización y de funcionamiento del indicador. Este parámetro acepta los valores 0, 2, 3, 16, 32 y 128. Estos valores pueden acumularse con el fin de definir varias opciones (excepto para 128). Estos son los detalles para cada valor:*
  - modo = 0: no mostrar las unidades.
  - modo = 2: mostrar las unidades a la derecha o debajo del indicador.
  - modo = 3: mostrar las unidades a la izquierda o sobre el indicador.
  - modo = 16: mostrar las graduaciones junto a las unidades.
  *- modo = 32: **On Data Change** se ejecuta mientras el usuario está ajustando el indicador. Si este valor no se utiliza, **On Data Change** ocurre sólo después que el usuario termina de ajustar el indicador.*
  - modo = 128: activar el modo "Barber shop (animación continua). Este valor no puede combinarse con otros. En este modo, se ignoran los otros parámetros (excepto para el parámetro display, si se pasa). Para mayor información sobre este modo, consulte el manual de Diseño.
- *format = formato de salida de las graduaciones del indicador.*
- *visualización = opciones de visualización específicas. En el caso de los termómetros, este parámetro sólo se tiene en cuenta cuando el subparámetro modo es igual a 128.*
  - *visualización = 0 (o se omite): mostrar una regla estándar/termómetro en animación continua "barber shop".*
  - *visualización = 1 : activa el modo "Stepper" para una regla / activa el modo "Progresión asincrónica" para un termómetro. Para mayor información sobre estas opciones consulte el manual de Diseño.*

Dials

Para dar formato a objetos de tipo dial, en el parámetro formatSalida, pase una cadena de caracteres respetando la siguiente sintaxis:
min;max;unid;interv{;modo}

- *min = valor de la primera graduación del indicador.*
- *max = valor de la última graduación del indicador.*
- *unid = intervalo entre las graduaciones del indicador.*
- *interv = intervalo mínimo del cursor de movimiento en el indicador.*
- *modo = modo de funcionamiento del dial (opcional). Este parámetro sólo acepta el valor 32: **On Data Change** se ejecuta mientras el usuario está ajustando el indicador. Si este valor no se utiliza, **On Data Change** ocurre sólo después de que el usuario haya terminado de ajustar el indicador.*

Rejillas de botones

Para dar formato a rejillas de botones, pase en el parámetro formatSalida una cadena de caracteres respetando la siguiente sintaxis:
cols;lines

- *cols = número de columnas de la rejilla.*
- *lineas = número de líneas de la rejilla.*

***Nota:** para mayor información sobre formatos de salida de los objetos de formulario, consulte el Manual de Diseño.*

Botones 3D

Para dar formato a botones 3D, pase en el parámetro formatSalida una cadena de caracteres respetando la siguiente sintaxis:
titulo;imagen;fondo;tituloPos;tituloVisible;iconVisible;estilo;horMargen;vertMargen;
iconOffset;popupMenu,hipervínculo;numEstados

- *titulo = título del botón. Este valor puede expresarse como texto o un número de recurso (ej.: ":16800,1")*
- *imagen = imagen asociada al botón, proveniente de la librería de imágenes, de una variable imagen o de un archivo imagen:*
  - Si la imagen proviene de una librería de imágenes, introduzca su número, precedido por un signo de interrogación (ej.: "?250").
  - Si la imagen proviene de una variable imagen, introduzca el nombre de la variable.
  - Si la imagen proviene de un archivo almacenado en el archivo Recursos de la base, introduzca un URL del tipo carpeta/}nomimagen" o "archivo:{carpeta/}nomimagen".
- *fondo= imagen de fondo asociada a un botón (estilo personalizado), que proviene de una librería de imágenes, de una variable imagen o de un archivo almacenado en la carpeta Recursos (ver arriba).*
- *tituloPos= posición del título del botón. Son posibles cinco valores:*
  - tituloPos = 0: Centro
  - tituloPos = 1: Derecha
  - tituloPos = 2: Izquierda
  - tituloPos = 3: Abajo
  - tituloPos = 4: Arriba
- *tituloVisible= Define si el título es visible o no. Dos valores son posibles:*
  - tituloVisible= 0: el título está oculto
  - tituloVisible= 1: el título se muestra
- *iconVisible = Define si el icono es visible o no. Son posibles dos valores:*
  - iconVisible = 0 : el icono está oculto
  - iconVisible = 1 : el icono se muestra
- *estilo = Estilo de botón. El valor de esta opción determina si otras opciones son tenidas en cuenta (por ejemplo, fondo). Los siguientes valores de estilo son posibles:*
  - estilo = 0: Ninguno
  - estilo = 1: Fondo desplazado
  - estilo = 2: Pulsador
  - estilo = 3: Botón de barra
  - estilo = 4: Personalizado
  - estilo = 5: Círculo
  - estilo = 6: Cuadrito de sistema
  - estilo = 7: Office XP
  - estilo = 8: Bevel
  - estilo = 9: Bevel redondeado
  - estilo = 10: Contraer/Expandir
  - estilo = 11: Ayuda
  - estilo = 12: OS X Texturizado
  - estilo = 13: OS X Gradiente
- *horMargen = Margen horizontal. Número de píxeles delimitando las márgenes internas a la derecha y a la izquierda del botón (áreas que el icono y el texto no deben invadir).*
- *vertMargen = Margen vertical. Número de píxeles delimitando las márgenes superior e inferior del botón (áreas que el icono y el texto no deben invadir).*
- *iconOffset = Desplazamiento del icono a la derecha y hacia abajo. Este valor, expresado en píxeles, indica la diferencia del icono del botón a la derecha y hacia abajo en caso de clic (el mismo valor se utiliza para ambas direcciones).*
- *popupMenu = Asociación de un menú desplegable con el botón. Son posibles tres valores:*
  - popupMenu = 0: Sin menú desplegable
  - popupMenu = 1: Con menú desplegable asociado
  - popupMenu = 2: Con menú desplegable separado
- *hipervínculo = El título es subrayado al pasar el cursor del ratón para parecerse a un hipervínculo (mecanismo obsoleto). Dos valores son posibles:*
  - *hipervínculo = 0: el título no es subrayado al pasar el ratón*
  - *hipervínculo = 1: el título es subrayado al pasar el ratón*
- *numEstados = Número de estados presentes en la imagen utilizada como icono para el botón 3D, y que serán utilizados por 4D para representar los estados de los botones estándar (de 0 a 4).*

Algunas opciones no se tienen en cuenta para todos los estilos de botones 3D. Adicionalmente, en algunos casos, usted podría querer no modificar todas las opciones. Para no pasar una opción, simplemente omita el valor correspondiente. Por ejemplo, si no quiere pasar las opciones titleVisible y vertMargen, puede escribir:

```
OBJECT SET FORMAT(miVar;"BonitoBotón;?256;:562;1;;1;4;5;;5;0")
```

Encabezados de list box

Para formatear el icono en un encabezado de list box, pase una cadena de caracteres en el parámetro formatSalida, que respeta la siguiente sintaxis:
imagen;iconPos

- *imagen= imagen de encabezado, procedente de la librería de imágenes, una variable imagen o un archivo imagen:*
- - *Si la imagen viene de la librería de imágenes, introduzca su número, precedido por un signo de interrogación (por ejemplo: "?250").*
  - *Si viene de una variable imagen, introduzca el nombre de la variable.*
  - *Si viene de un archivo almacenado en la carpeta de recursos de la base, introduzca una URL del tipo "# {carpeta/}nombreimagen" o "file: {carpeta/}nombreimagen".*
- *iconPos = posición del icono en el encabezado. Dos valores son soportados:*
  - *iconPos = 1: Izquierda*
  - *iconPos = 2: Derecha*

Esta funcionalidad es útil, por ejemplo, cuando se quiere trabajar con un icono personalizado.

Ejemplo 1

La siguiente línea de código da formato al campo [Empleados]Fecha Contratado al quinto formato de fecha (Internal date long).

```
OBJECT SET FORMAT([Empleados]Fecha Contratado;Char(Internal date long))
```

Ejemplo 2

El siguiente ejemplo cambia el formato de un campo [Empresa]Codigo postal de acuerdo con la longitud del valor en el campo:

```
If(Length([Empresa]Codigo postal)=9)
   OBJECT SET FORMAT([Empresa]Codigo postal;"#####-####")
Else
   OBJECT SET FORMAT([Empresa]Codigo postal;"#####")
End if
```

Ejemplo 3

El siguiente ejemplo da formato al valor de un campo numérico dependiendo de si es positivo, negativo, o nulo:

```
OBJECT SET FORMAT([Stats]Results;"### ##0.00;(### ##0.00);")
```

Ejemplo 4

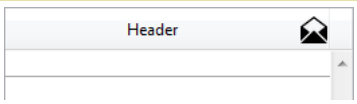
El siguiente ejemplo define el formato de un campo Booleano para mostrar Casado y Soltero, en lugar de los valores por defecto Sí y No:

```
OBJECT SET FORMAT([Empleado]Estado Civil;"Casado;Soltero")
```

Ejemplo 5

Siempre que haya almacenado un archivo de imagen llamado "envelope_open.png" en la carpeta Resources de la base, puede escribir:

```
vIcon:="#envelope_open.png"
vPos:="2" // Right
OBJECT SET FORMAT(*;"Header1";vIcon+";"+vPos)
```



Ejemplo 6

El siguiente ejemplo define el formato de un campo booleano para mostrar una casilla de selección llamada "Clasificado":

```
OBJECT SET FORMAT([Carpeta]Clasificación;"Clasificado")
```

Ejemplo 7

Usted tiene una tabla de miniaturas que contiene 1 línea y 4 columnas, destinada a mostrar un botón imagen ("activa por defecto", "al hacer clic en el botón", "al pasar el cursor" e "inactivo"). Usted quiere asociar las opciones Cambiar al pasar el cursor encima, Volver al soltar el clic y Última imagen si desactivado:

```
OBJECT SET FORMAT(*;"BotonImagen";"4;1;?15000;176")
```

Ejemplo 8

Pase un termómetro a modo "Barber shop"

```
OBJECT SET FORMAT($Mitermo;";;;;128")
$Mitermo :=1  `Iniciar la animación
```

OBJECT SET HELP TIP

OBJECT SET HELP TIP ({* ;} objeto ; mensajeAyuda)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
mensajeAyuda	Texto	➡	Contenido del mensaje de ayuda

Descripción

El comando ***OBJECT SET HELP TIP*** *permite definir o modificar dinámicamente el mensaje de ayuda asociado al objeto o los objetos designados por los parámetros objeto y * para el proceso actual.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

En el parámetro mensajeAyuda, pase una cadena de caracteres para el contenido del mensaje. Si pasa una cadena vacía "", la sugerencia de ayuda se eliminará.

Cuando se ejecuta el formulario, los mensajes aparecen como mensajes de ayuda cuando el cursor se mueve sobre el campo u objeto. El retardo de visualización y la duración máxima de los mensajes de ayuda se pueden controlar utilizando los selectores Tips delay y Tips duration del comando ***SET DATABASE PARAMETER****.*

Pase el contenido del mensaje en el parámetro mensajeAyuda. Puede pasar:

- *una cadena de caracteres, por ejemplo "Utilice el / como separador",*
- *una cadena vacía "" para eliminar el mensaje de ayuda,*

Cuando el formulario se ejecuta, los mensajes de ayuda aparecen como mensajes de ayuda cuando el cursor pasa sobre el campo u objeto. El retraso de visualización y la duración máxima de los mensajes de ayuda pueden controlarse utilizando Tips delay y los selectores del comando ***SET DATABASE PARAMETER****.*

Puede utilizar este comando con un objeto list box para agregar mensajes de ayuda a las filas y celdas del list box. Por ejemplo, un objeto list box puede tener un mensaje de ayuda diferente por fila. En este caso, primero debe determinar la posición del cursor con el comando ***LISTBOX GET CELL POSITION****. Esto se muestra en un ejemplo a continuación.*

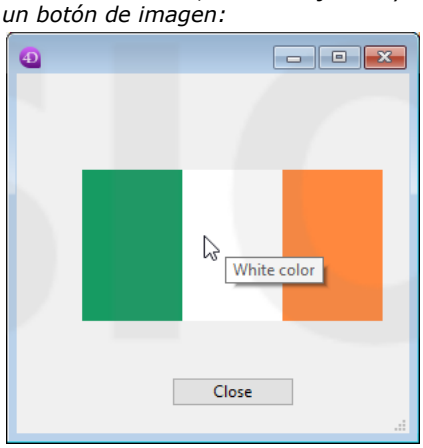
Cuando aparece un mensaje de ayuda, llamando al comando ***OBJECT SET HELP TIP*** *se cierra, abre un nuevo mensaje en la ubicación del ratón y reinicia el contador Tips duration, permitiendo el manejo dinámico de las sugerencias.*

Notas:

- *El contenido del mensaje de ayuda también se puede definir utilizando el editor de formularios (ver* ***Controles y ayuda a la entrada****) y el editor de estructura (ver* ***Propiedades de los campos****) en modo Diseño.*
- *Los consejos de ayuda pueden desactivarse globalmente para la aplicación utilizando el selector Tips enabled del comando* ***SET DATABASE PARAMETER****.*

Ejemplo 1

En este formulario, un mensaje de ayuda se visualiza y cambia dinámicamente cuando el ratón pasa sobre diferentes partes de un botón de imagen:




```
//Método objeto "myFlag"

C_REAL($x;$y;oldX;oldY)
C_REAL($left;$right;$top;$bottom)
C_LONGINT($b)
C_TEXT($tip)
C_TEXT(oldTip)
C_BOOLEAN($doRefresh)
```

```
Case of
   :(Form event=On Load)
      oldTip:=""
      SET DATABASE PARAMETER(Tips enabled;1) //Para asegurarse de que los consejos están habilitados
      SET DATABASE PARAMETER(Tips delay;0) // Sugerencia mostrada inmediatamente al detener el ratón
      SET DATABASE PARAMETER(Tips duration;60*10) // 10 segundos máximo de visualización
   :(Form event=On Mouse Move)
      GET MOUSE($x;$y;$b)
      OBJECT GET COORDINATES(*;"myFlag";$left;$top;$right;$bottom)
      $x:=$x-$left
      $y:=$y-$top
      Case of  //cada parte de la bander es de 76 píxeles
         :($x<76)
            $tip:="Green color"
         :($x<152)
            $tip:="White color"
         Else
            $tip:="Orange color"
      End case

      $doRefresh:=($tip#oldtip) //true si el mensaje es diferente
      If(Not($doRefresh)) //Los mismos contenidos
         $doRefresh:=((Abs($x-oldX)>30)|(Abs($y-oldY)>30)) //true si se mueve el cursor
      End if

      If($doRefresh) //mostrar otro mensaje
         OBJECT SET HELP TIP(*;"myFlag";$tip)
         oldX:=$x
         oldY:=$y
         oldTip:=$tip
      End if

End case
```

Ejemplo 2

Usted tiene un list box, "Commands List", que contiene una lista y desea definir un mensaje de ayuda que muestre la descripción de cada elemento de lista. La descripción se encuentra en la tabla [Documentation].

```
C_REAL($mouseX;$mouseY;$mouseZ)
C_LONGINT($col;$row)

Case of

   :(Form event=On Mouse Enter)

      SET DATABASE PARAMETER(Tips delay;1) // hace que el mensaje aparezca rápidamente

   :(Form event=On Mouse Move)

  //#1 : encuentra la fila que se ha movido

      GET MOUSE($mouseX;$mouseY;$mouseZ)
      LISTBOX GET CELL POSITION(*;"Commands List";$mouseX;$mouseY;$col;$row)

  //#2 : configure el mensaje de ayuda correspondiente

      If($row#0)
         GOTO SELECTED RECORD([Documentation];$row)
         OBJECT SET HELP TIP(*;"Commands List";[Documentation]Description) // La descripción completa se utilizará como "mensaje de ayuda" cuando (si) el ratón deja de moverse.
      End if

   :(Form event=On Mouse Leave)

      SET DATABASE PARAMETER(Tips delay;3) // hace que el mensaje aparezca normalmente

End case
```

El resultado es...

Commands and functions	URL
ABORT	http://doc.4d.com/4Dv16/4D/16.1/ABORT.301-3374748.en.html
ASSERT	http://doc.4d.com/4Dv16/4D/16.1/ASSERT.301-3374754.en.html
ON ERR CALL	http://doc.4d.com/4Dv16/4D/16.1/ON-ERR-CALL.301-3374749.en.html
SET ASSERT ENABLED	http://doc.4d.com/4Dv16/4D/16.1/SET-ASSERT-ENABLED.301-3374751.en.html
APPEND TO ARRAY	http://doc.4d.com/4Dv16/4D/16.1/APPEND-TO-ARRAY.301-3375651.en.html
ARRAY REAL	http://doc.4d.com/4Dv16/4D/16.1/ARRAY-REAL.301-3375671.en.html
LIST TO ARRAY	679.en.html
DELETE FROM ARR	-3375646.en.html
LIST TO ARRAY	http://doc.4d.com/4Dv16/4D/16.1/LIST-TO-ARRAY.301-3375679.en.html
SORT ARRAY	http://doc.4d.com/4Dv16/4D/16.1/SORT-ARRAY.301-3375667.en.html

The ARRAY REAL command creates and/or resizes an array of Real elements in memory.

OBJECT SET HORIZONTAL ALIGNMENT

OBJECT SET HORIZONTAL ALIGNMENT ({* ;} objeto ; alineación)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre del objeto (cadena) Si se omite= objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
alineación	Entero largo	➡	Código de alineación

Descripción

El comando **OBJECT SET HORIZONTAL ALIGNMENT** permite determinar el tipo de alineación aplicado al objeto o a los objetos designados por los parámetros objeto y *.

Si especifica el parámetro opcional *, indica que el parámetro objeto designa el nombre de un objeto (una cadena). Si no pasa el parámetro *, indica que el parámetro objeto designa un campo o una variable. En este caso, usted no pasa una cadena sino la referencia de un campo o de una variable (campo o variable de tipo objeto únicamente).

Pase en el parámetro alineación una de las constantes del tema **Propiedades de los objetos**:

Constante	Tipo	Valor	Comentario
Align center	Entero largo	3	
Align default	Entero largo	1	
Align left	Entero largo	2	
Align right	Entero largo	4	
wk justify	Entero largo	5	Disponible para áreas 4D Write Pro únicamente

Nota: la constante wk justify está disponible en el thema "**4D Write Pro**".

Los objetos de formulario a los cuales puede aplicar este comando son los siguientes:

- Áreas de desplazamiento
- Combo boxes
- Textos estáticos
- Áreas de grupos
- Menús desplegables/Listas desplegables
- Campos
- Variables
- List boxes
- Columnas de list box
- Encabezados de list box
- Pies de list box
- Áreas **Referencia 4D Write Pro**

OBJECT SET INDICATOR TYPE

OBJECT SET INDICATOR TYPE ({* ;} objeto ; indicador)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
indicador	Entero largo	➡	Tipo de indicador

Descripción

El comando ***OBJECT SET INDICATOR TYPE*** *modifica el tipo de indicador de progresión del o de los termómetro(s) designado(s) por los parámetros objeto y * en el proceso actual.*

El tipo de indicador define la variante de visualización del termómetro. Para más información, consulte ***Indicadores*** *en el manual de Diseño.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*En el parámetro indicador, pase el tipo de indicador a mostrar. Puede utilizar una de las siguientes constantes del tema "****Propiedades de los objetos****":*

Constante	Tipo	Valor	Comentario
Asynchronous progress bar	Entero largo	3	Indicador circular muestra una animación continua
Barber shop	Entero largo	2	Barra que muestra una animación continua
Progress bar	Entero largo	1	Barra de progreso estándar

OBJECT SET KEYBOARD LAYOUT

OBJECT SET KEYBOARD LAYOUT ({* ;} objeto ; codigoLeng)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
codigoLeng	Cadena	➡	Código de lenguaje RFC3066 ISO639 e ISO3166, "" = no cambiar

Descripción

El comando ***OBJECT SET KEYBOARD LAYOUT*** *permite definir o modificar dinámicamente la configuración de teclado asociada al objeto o los objetos designados por los parámetros objeto y * para el proceso actual.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el objeto es una variable o un campo. En este caso, pase una referencia en lugar de un nombre.*

En codigoLeng, pase una cadena indicando el código del lenguaje a utilizar, basado en RFC3066, ISO639 e ISO3166. Para mayor información, consulte la descripción del comando ***SET DATABASE LOCALIZATION****.*

OBJECT SET LIST BY NAME

OBJECT SET LIST BY NAME ({* ;} objeto {; listType}; lista)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
listType	Entero largo	➡	Tipo de lista: Lista de selección, Lista obligatoria o Lista de excluidos
lista	Cadena	➡	Nombre de la lista a utilizar (definida en el entorno Diseño)

Descripción

*El comando **OBJECT SET LIST BY NAME** define, reemplaza o disocia la lista asociada al objeto o al grupo de objetos designado por objeto.La lista cuyo nombre se pasa en el parámetro lista debe haber sido creada en el editor de listas, en modo Diseño.*

Este comando puede aplicarse a un formulario de entrada o diálogo, a campos y variables editables cuyos valores pueden introducirse como texto.

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, usted especifica una referencia de un campo o de una variable (campo o variable de tipo objeto únicamente) en lugar de una cadena.*

***Nota:** este comando no puede utilizarse con campos ubicados en un formulario listado de un subformulario.*
*El comando **OBJECT SET LIST BY NAME** le permite definir o remplazar todos los tipos de listas asociadas a los objetos designados por los parámetros objeto y *: listas de selección, lista de valores obligatorios y listas de valores excluidos. Para ello, en el parámetro tipoLista pase una de las siguientes constantes, que se encuentra en el tema "**Propiedades de los objetos**":*

Constante	Tipo	Valor	Comentario
Choice list	Entero largo	0	Lista simple de selección de valores (opción "Lista" en la Lista de Propiedades) (por defecto)
Excluded list	Entero largo	2	Lista de valores no aceptados para la entrada (Opción "Exclusiones" en la lista de propiedades)
Required list	Entero largo	1	Lista sólo los valores aceptados para la entrada (Opción "Obligatoria" en la Lista de Propiedades)

Si omite este parámetro, el valor 0 (lista de selección) se utiliza por defecto.

En el proceso actual, para desvincular una lista que se asoció al objeto, pase una cadena vacía ("") en el parámetro lista para el tipo de lista concerniente.

Ejemplo 1

El siguiente ejemplo define una lista asociada a un campo Envío. Si el envío debe realizarse en la noche, entonces la lista muestra las empresas que realizan envíos en la noche. De lo contrario, se asignan las empresas de envíos estándar:

```
If([Envios]Overnight)
    <p>OBJECT SET LIST BY NAME([Envios]Empresa;"Envios de noche")
Else
    OBJECT SET LIST BY NAME([Envios]Empresa;"Envios estándar")
End if
```

Ejemplo 2

Asocia la lista "color_choice" como una lista desplegable simple llamada "DoorColor":

```
OBJECT SET LIST BY NAME(*;"DoorColor";Choice list;"color_choice")
 // en este caso, el tercer parámetro (constante) puede omitirse
```

Ejemplo 3

Usted desea asociar la lista "color_choice" al combo box "WallColor". Como este combo box es editable, usted desea que no sea posible usar ciertos colores como el "negro", "morado", etc. Estos colores se colocan en la lista "excl_colors":

```
OBJECT SET LIST BY NAME(*;"WallColor";Choice list;"color_choice")
OBJECT SET LIST BY NAME(*;"WallColor";Excluded list;"excl_colors")
```

Ejemplo 4

Usted quiere eliminar la lista de asociaciones:

```
 // eliminación de una lista de selección
OBJECT SET LIST BY NAME(*;"DoorColor";Choice list;"")
 // eliminación de una lista de valores que que no son permitidos
OBJECT SET LIST BY NAME(*;"WallColor";Excluded list;"")
```

OBJECT SET LIST BY REFERENCE

OBJECT SET LIST BY REFERENCE ({* ;} objeto {; tipoLista}; lista)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
tipoLista	Entero largo	➡	Tipo de lista: Lista de valores, Lista de obligatorios o Lista de excluidos
lista	ListRef	➡	Número de referencia de lista

Descripción

*El comando **OBJECT SET LIST BY REFERENCE** define o remplaza la lista asociada con el objeto u objetos definidos por los parámetros objeto y *, con la lista jerárquica definida en el parámetro lista.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*Por defecto, si se omite el parámetro tipoLista, el comando define una lista de selección fuente (selección de valores) para el objeto. Puede designar cualquier tipo de lista en el parámetro tipoLista. Para ello, sólo tiene que pasar una de las siguientes constantes del tema "**Propiedades de los objetos**":*

Constante	Tipo	Valor	Comentario
Choice list	Entero largo	0	Lista simple de selección de valores (opción "Lista" en la Lista de Propiedades) (por defecto)
Excluded list	Entero largo	2	Lista de valores no aceptados para la entrada (Opción "Exclusiones" en la lista de propiedades)
Required list	Entero largo	1	Lista sólo los valores aceptados para la entrada (Opción "Obligatoria" en la Lista de Propiedades)

*En lista, pase el número de referencia de la lista jerárquica que desea asociar al objeto. Esta lista debe haber sido generada utilizando el comando **Copy list**, **Load list** o **New list**.*

Para finalizar la asociación de una lista con un objeto, sólo pase 0 en el parámetro lista para el tipo de lista concerniente.

*Eliminar una asociación de lista, no elimina la referencia de lista en memoria. Recuerde llamar el comando **CLEAR LIST** cuando ya no necesite la lista.*

Este comando es especialmente interesante en el contexto de un pop-up o combo box asociado a una variable o un campo (ver el Manual de Diseño). En este caso, la asociación es dinámica y cualquier cambio en la lista se copia en el formulario. Cuando el objeto está asociado a un array, la lista se copia en el array y cualquier cambio en la lista no están disponible de forma automática (ver el ejemplo 5).

Ejemplo 1

Asociar una lista de opciones simples (tipo de lista predeterminado) a un campo de texto:

```
vCountriesList:=New list
APPEND TO LIST(vCountriesList;"Spain";1)
APPEND TO LIST(vCountriesList;"Portugal";2)
APPEND TO LIST(vCountriesList;"Greece";3)
OBJECT SET LIST BY REFERENCE([Contact]Country;vCountriesList)
```

Ejemplo 2

Asociar la lista "vColor" como una lista de selección simple con el pop-up/lista desplegable "DoorColor":

```
vColor:=New list
APPEND TO LIST(vColor;"Blue";1)
APPEND TO LIST(vColor;"Green";2)
APPEND TO LIST(vColor;"Red";3)
APPEND TO LIST(vColor;"Yellow";4)
OBJECT SET LIST BY REFERENCE(*;"DoorColor";Choice list;vColor)
```

Ejemplo 3

Ahora desea asociar la lista "vColor" con un combo box denominado "WallColor". Como este combo box es editable, usted quiere asegurarse de que ciertos colores, como "negro", "morado", etc, no se puedan utilizar. Estos colores se colocan en la lista

"vReject":

```
OBJECT SET LIST BY REFERENCE(*;"WallColor";Choice list;vColor)
vReject:=New list
APPEND TO LIST(vReject;"Black";1)
APPEND TO LIST(vReject;"Gray";2)
APPEND TO LIST(vReject;"Purple";3)
OBJECT SET LIST BY REFERENCE(*;"WallColor";Excluded list;vReject)
```

Ejemplo 4

Usted desea eliminar las asociaciones de listas:

```
OBJECT SET LIST BY REFERENCE(*;"WallColor";Choice list;0)
OBJECT SET LIST BY REFERENCE(*;"WallColor";Required list;0)
OBJECT SET LIST BY REFERENCE(*;"WallColor";Excluded list;0)
```

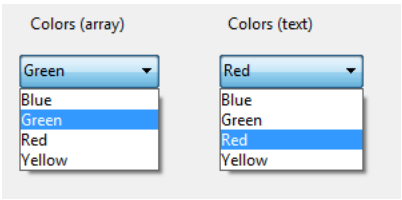
Ejemplo 5

Este ejemplo ilustra la diferencia en la forma en que funciona el comando cuando se aplica a un menú pop-up asociado a un array texto o a una variable texto. Hay dos menús pop-up en un formulario:

El contenido de estos menús emergentes se define utilizando la lista <>vColor (que contiene los valores de colores). Se ejecuta el siguiente código cuando se carga el formulario:

```
ARRAY TEXT(arr1;0) //arr1 pop up
C_TEXT(text1) //text1 pop up
OBJECT SET LIST BY REFERENCE(*;"arr1";<>vColor)
OBJECT SET LIST BY REFERENCE(*;"text1";<>vColor)
```

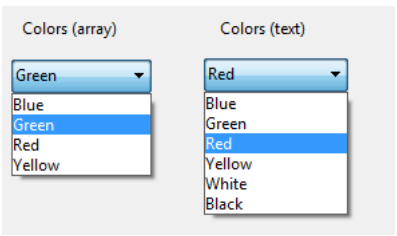
Durante la ejecución, ambos menús proponen los mismos valores::



(Montaje que muestra el contenido de los menús de forma simultánea)
Luego ejecute el siguiente código, por ejemplo, por medio de un botón:

```
APPEND TO LIST(<>vColor;"White";5)
APPEND TO LIST(<>vColor;"Black";6)
```

Sólo el menú asociado al campo texto se actualiza (por medio de la referencia dinámica):



Con el fin de actualizar la lista asociada al pop-up gestionado por array, es necesario llamar de nuevo al comando ***OBJECT SET LIST BY REFERENCE*** *para copiar el contenido de la lista.*

OBJECT SET MAXIMUM VALUE

OBJECT SET MAXIMUM VALUE ({* ;} objeto ; valorMax)

Parámetro	*Tipo*		*Descripción*
*	*Operador*	➡	*Si especifica, objeto es un nombre de objeto(cadena) Si se omite, objeto es un campo o variable*
objeto	*Objeto de formulario*	➡	*Nombre del objeto (si * se especifica) o Campo o variable (si * se omite)*
valorMax	*Fecha, Hora, Número*	➡	*Valor máximo para el objeto*

Descripción

*El comando **OBJECT SET MAXIMUM VALUE** modifica el valor máximo del objeto o de los objetos designado(s) por los parámetros objeto y * para el proceso actual.*

*La propiedad "Valor máximo" se puede aplicar a datos de tipo número, fecha u hora. Para más información, consulte **Valores máximos y mínimos** en el manual de Diseño.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

En valorMax, pase el nuevo valor máximo que desea asignar al objeto para el proceso actual. Este valor debe coincidir con el tipo de objeto, de lo contrario se devuelve el error 18 "Tipos incompatibles".

OBJECT SET MINIMUM VALUE

OBJECT SET MINIMUM VALUE ({* ;} objeto ; valorMinimo)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
valorMinimo	Fecha, Hora, Número	➡	Valor mínimo para el objeto

Descripción

*El comando **OBJECT SET MINIMUM VALUE** modifica el valor mínimo del objeto o de los objetos designado(s) por los parámetros objeto y * para el proceso actual.*

*La propiedad "valorMinimo" se puede aplicar a datos de tipo número, fecha u hora. Para más información, consulte **Valores máximos y mínimos** en el manual de Diseño.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

En valorMinimo, pase el nuevo valor mínimo a asignar al objeto para el proceso actual. Este valor debe coincidir con el tipo de objeto, de lo contrario se muestra el mensaje error 18 "Los tipos de campo son incompatibles".

OBJECT SET MULTILINE

OBJECT SET MULTILINE ({* ;} objeto ; multilinea)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
multilinea	Entero largo	➡	Estado de la propiedad multilínea

Descripción

El comando ***OBJECT SET MULTILINE*** *modifica la propiedad "Multilínea" del objeto(s) designada por los parámetros objeto y * .*

La propiedad "Multilínea" controla dos parámetros relacionados con la visualización e impresión de las áreas de texto: visualización de palabras situadas al final de la línea en las áreas de una sola línea y la inserción automática de los retornos de línea. Para obtener más información, consulte ***Multilíneas*** *en el manual de Diseño. Si se aplica este comando a un objeto que no admite esta propiedad, el comando no hace nada.*

*Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable de objeto únicamente).*

*En el parámetro multilinea, pase el nuevo valor de la opción que desea ajustar. Puede utilizar las siguientes constantes, que se encuentran en el tema "****Propiedades de los objetos****":*

Constante	Tipo	Valor	Comentario
Multiline Auto	Entero largo	0	En las áreas de una sola línea, las palabras situadas al final de las líneas se cortan y no hay retornos de línea. En las áreas de varias líneas, 4D efectúa saltos de línea automáticos.
Multiline No	Entero largo	2	Nunca hay vuelta de la línea: el texto se muestra siempre en una sola línea. Si el campo o la variable alfa o texto contiene retornos de carro, el texto situado después del primer retorno de carro se elimina tan pronto como se modifica el área.
Multiline Yes	Entero largo	1	En las áreas de una sola línea, el texto se muestra hasta el primer retorno de carro o hasta la última palabra que se puede mostrar por completo. 4D inserta retornos de línea, es posible desplazarse por el contenido del área con la tecla de flecha hacia abajo. En las áreas de varias líneas, 4D efectúa los saltos de línea automáticos.

Ejemplo

Usted quiere prohibir varias líneas en un área de entrada:

```
OBJECT SET MULTILINE(*;"vEntry";Multiline No)
```

OBJECT SET PLACEHOLDER

OBJECT SET PLACEHOLDER ({* ;} objeto ; textoEjemplo)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
textoEjemplo	Texto	➡	Texto de ejemplo asociado al objeto

Descripción

El comando ***OBJECT SET PLACEHOLDER*** *asocia un texto de ejemplo al objeto o a los objetos designados por los parámetros objeto y * .*

Para obtener más información sobre los textos de ejemplo, consulte el manual de Diseño.

Si un texto de ejemplo ya está asociado al objeto vía la Lista de propiedades, este texto se sustituye en el proceso actual por el contenido del parámetro textoEjemplo.

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente) .*

En textoEjemplo, pase el texto de ayuda o la indicación que debe aparecer cuando el objeto esté vacío.

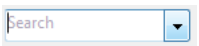
Nota**: el comando* ***OBJECT SET PLACEHOLDER *no soporta la inserción de referencias XLIFF en los textos de ejemplo. Esto sólo es posible para los textos de ejemplo definidos mediante la Lista de propiedades.*

Este comando sólo se puede utilizar con objetos de formulario de tipo variable, campo o combo box. También puede se asociar a los datos de tipo fecha u hora si el objeto de formulario tiene la propiedad "Vacío si nulo".

Ejemplo

Usted quiere mostrar el texto "Buscar" un combo box:

```
OBJECT SET PLACEHOLDER(*;"search_combo";"Search")
```



OBJECT SET PRINT VARIABLE FRAME

OBJECT SET PRINT VARIABLE FRAME ({* ;} objeto ; marcoVariable {; subformFijo})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
marcoVariable	Booleano	➡	True = Impresión de marco variable, False = Impresión de marco fijo
subformFijo	Entero largo	➡	Opciones de impresión de subformularios en tamaño fijo

Descripción

*El comando **OBJECT SET PRINT VARIABLE FRAME** modifica la propiedad de marco de impresión variable del objeto o de los objetos designados por los parámetros objeto y *.*

Esta propiedad está disponible para los siguientes objetos:

- *variables y campos de tipo Texto e Imagen (ver **Impresión tamaño variable** en el manual de Diseño)*
- *áreas 4D Write Pro (ver **Utilizar un área 4D Write Pro** en el manual 4D Write Pro).*
- *Subformularios. Los subformularios tienen una opción adicional para la impresión de tamaño fijo (ver **Impresión** en el manual de Diseño); el comando puede ser utilizado para configurar esta opción utilizando el parámetro subformFijo.*

Si aplica este comando a un objeto que no soporta esta propiedad, el comando no hace nada.

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

*Pase un booleano en el parámetro marcoVariable: si pasa **True**, el objeto se imprime con un marco variable. Si pasa **False**, se imprime con un marco fijo.*

*El parámetro opcional subformFijo le permite configurar una opción adicional cuando pasa **False** en el parámetro marcoVariable y el objeto es un subformulario (se ignora en todos los demás casos). En este caso, se puede definir el modo de impresión de marco fijo para el subformulario. Puede pasar una de las siguientes constantes, que se encuentran en el tema "**Propiedades de los objetos**":*

Constante	Tipo	Valor	Comentario
Print Frame fixed with multiple records	Entero largo	2	El tamaño inicial del marco permanece del mismo tamaño, 4D imprime el formulario varias veces para incluir todos los registros.
Print Frame fixed with truncation	Entero largo	1	4D imprime sólo los registros que aparecen en el área del subformulario. El formulario se imprime sólo una vez y los registros que no se imprimen se ignoran.

OBJECT SET RESIZING OPTIONS

OBJECT SET RESIZING OPTIONS ({* ;} objeto ; horizontal ; vertical)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
horizontal	Entero largo	➡	Opción de redimensionamiento horizontal
vertical	Entero largo	➡	Opción de redimensionamiento vertical

Descripción

*El comando **OBJECT SET RESIZING OPTIONS** permite definir o modificar dinámicamente las opciones de redimensionamiento del objeto o de los objetos designados por los parámetros objeto y * para el proceso actual. Estas opciones definen la visualización del objeto en caso de redimensionamiento de la ventana del formulario.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

*En el parámetro horizontal, pase un valor que indique la opción de redimensionamiento horizontal a definir para el objeto. Puede pasar una de las siguientes constantes, del tema **Propiedades de los objetos**:*

Constante	Tipo	Valor	Comentario
Resize horizontal grow	Entero largo	1	Si la ventana se agranda un 50% de ancho, el objeto se agranda 50% a la derecha
Resize horizontal move	Entero largo	2	Si la ventana se agranda 100 píxeles de ancho, el objeto se mueve 100 píxeles a la derecha
Resize horizontal none	Entero largo	0	Si la ventana se agranda de ancho, ni el largo ni la posición del objeto varían

*En el parámetro vertical, pase un valor que indique la opción de redimensionamiento vertical a definir para el objeto. Puede pasar una de las siguientes constantes, del tema **Propiedades de los objetos**:*

Constante	Tipo	Valor	Comentario
Resize vertical grow	Entero largo	1	Si la ventana se agranda un 50% de alto, el objeto se alarga 50% hacia abajo
Resize vertical move	Entero largo	2	Si la ventana se agranda 100 píxeles de alto, el objeto se mueve 100 píxeles hacia abajo
Resize vertical none	Entero largo	0	Si la ventana se agranda de alto, ni el ancho ni la posición del objeto varían

OBJECT SET RGB COLORS

OBJECT SET RGB COLORS ({* ;} objeto ; colorPrimerPlano ; colorFondo {; colorFondoAlt})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
colorPrimerPlano	Entero largo	➡	Valor del color RGB del primer plano
colorFondo	Entero largo	➡	Valor del color RGB del fondo
colorFondoAlt	Entero largo	➡	Valor del color RGB del fondo alterno

Descripción

El comando OBJECT SET RGB COLORS cambia los colores de fondo y primer plano de los objetos especificados por objeto y el parámetro opcional *. Cuando el comando se aplica a un objeto list box, un parámetro adicional le permite modificar el color alterno de las filas.

Si pasa el parámetro opcional *, especifica que el parámetro objeto es el nombre de objeto (una cadena). Si omite el parámetro opcional *, especifica que objeto es un campo o una variable. En este caso, no pasa en objeto una cadena sino una referencia a un campo o a una variable (campo o variable de tipo objeto únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección **Propiedades de los objetos**.

El parámetro opcional colorFondoAlt le permite definir un color alternativo para el fondo de las líneas pares. Este parámetro sólo se utiliza cuando el objeto se especifica como List box o columna de List box. Cuando se utiliza este parámetro, el parámetro colorFondo se utiliza para las líneas impares únicamente. La utilización de colores alternativos hace que las listas sean más fáciles de leer.

Si objeto especifica un objeto List box, los colores alternos se utilizan en todo el List box. Si objeto especifica una columna del List box, sólo la columna utilizará los colores definidos.

Los valores de los colores se indican en RGB en colorPrimerPlano, colorFondo y colorFondoAlt. Un valor RGB es un entero largo de 4 bytes cuyo formato (0x00RRGGBB) se describe en la siguiente tabla (los bytes son numerados de 0 a 3, de derecha a izquierda):

Byte	Descripción
3	Debe ser cero para un color RGB absoluto
2	Componente rojo del color (0..255)
1	Componente verde del color (0..255)
0	Componente azul del color (0..255)

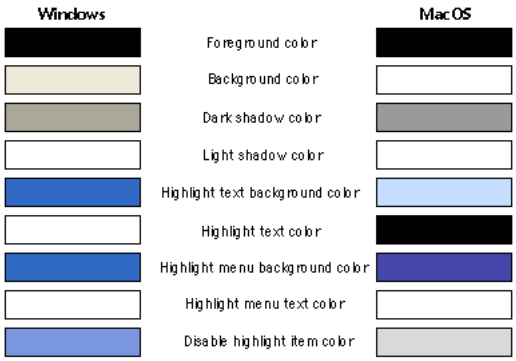
La siguiente tabla muestra algunos ejemplos de valores de color RGB:

Valor	Descripción
0x00000000	Negro
0x00FF0000	Rojo vivo
0x0000FF00	Verde vivo
0x000000FF	Azul vivo
0x007F7F7F	Gris
0x00FFFF00	Amarillo vivo
0x00FF7F7F	Rojo pastel
0x00FFFFFF	Blanco

También puede especificar uno de los colores "sistema" utilizado por defecto por 4D para dibujar los objetos cuya propiedad de color es "automática". Las siguientes constantes predefinidas son propuestas por 4D:

Constante	Tipo	Valor	Comentario
Background color none	Entero largo	-16	Esta constante puede ser utilizada únicamente con los parámetros colorFondo y colorFondoAlt.
Disable highlight item color	Entero largo	-11	
Highlight menu text color	Entero largo	-10	
Highlight menu background color	Entero largo	-9	
Highlight text color	Entero largo	-8	
Highlight text background color	Entero largo	-7	
Light shadow color	Entero largo	-4	
Dark shadow color	Entero largo	-3	
Background color	Entero largo	-2	
Foreground color	Entero largo	-1	

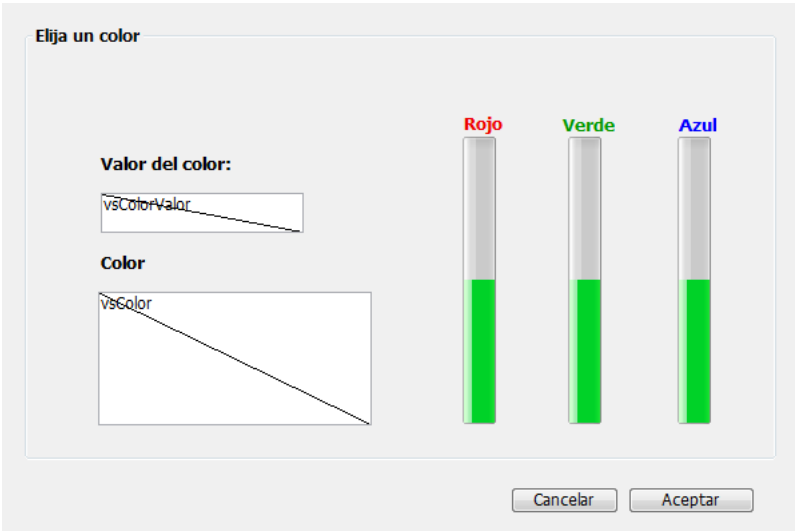
Por ejemplo, puede obtener los colores siguientes para los objetos de tipo campo o variable editables en los sistemas estándar:



Advertencia: *estos colores automáticos son dependientes del sistema y del tipo de objeto al cual afectan. En función de la versión del sistema operativo o si personaliza sus colores sistema, los colores automáticos de 4D se ajustarán en consecuencia. Utilice los valores de colores automáticos para asignar a los objetos los colores sistema y no para asignar los mismos colores que los definidos en el ejemplo anterior.*

Ejemplo 1

Este formulario contiene dos variables editables, vsColorValor y vsColor así como tres termómetros: thRojo, thVerde, y thAzul.



Estos son los métodos de estos objetos:

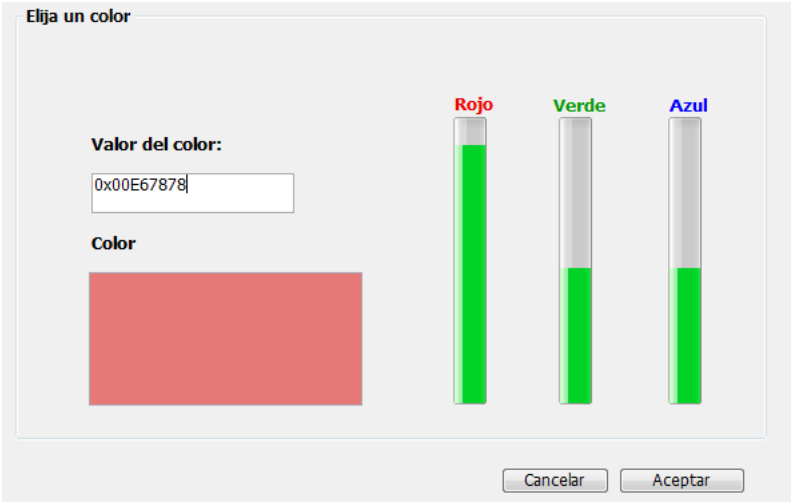
```
` Método de objeto de la variable no editable vsColorValor
Case of
   :(Form event=On Load)
      vsColorValor:="0x00000000"
End case

` Método de objeto de la variable no editable vsColor
Case of
   :(Form event=On Load)
      vsColor:=""
      OBJECT SET RGB COLORS(vsColor;0x00FFFFFF;0x0000)
End case

` Método de objeto del termómetro thRojo
CLIC EN TERMOMETRO DE COLOR
` Método de objeto del termómetro thVerde
CLIC EN TERMOMETRO DE COLOR

` Método de objeto del termómetro thAzul
CLIC EN TERMOMETRO DE COLOR
```

El método de proyecto llamado por los tres termómetros es:

```
` Método de proyecto CLIC EN TERMOMETRO DE COLOR
OBJECT SET RGB COLORS(vsColor;0x00FFFFFF;(thRojo << 16)+(thVerde << 8)+thAzul)
vsColorValor:=String((thRojo << 16)+(thVerde << 8)+thAzul;"&x")
If(thRojo=0)
   vsColorValor:=Substring(vsColorValor;1;2)+"0000"+Substring(vsColorValor;3)
End if
```

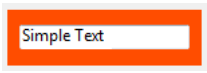
*Note la utilización de los **Bitwise operators** para el calculo de los valores de los colores a partir de los valores de los termómetros.*

En ejecución, el formulario se ve así:



Ejemplo 2

Cambia a fondo transparente con un color de fuente claro:




```
OBJECT SET RGB COLORS(*;"myVar";Light shadow color;Background color none)
```

OBJECT SET SCROLL POSITION

OBJECT SET SCROLL POSITION (* ; objeto {; posicionL {; posicionH}}{; *})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una tabla o una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Tabla o variable (si se omite *)
posicionL	Entero largo	➡	Número de línea a mostrar
posicionH	Entero largo	➡	Número de columna a mostrar (list box) o Desplazamiento horizontal en píxeles (imágenes)
*	Operador	➡	Mostrar la línea en primera posición después del desplazamiento

Descripción

*El comando **OBJECT SET SCROLL POSITION** permite desplazar el contenido de varios tipos de objetos: líneas de un subformulario, de un formulario listado mostrado utilizando los comandos **MODIFY SELECTION**, **DISPLAY SELECTION** o de una lista jerárquica, líneas y columnas de un list box o incluso los píxeles de una imagen.*

***Nota:** el desplazamiento por programación de un objeto continúa siendo posible incluso si las barras de desplazamiento están ocultas en el formulario.*

*Si pasa el primer parámetro opcional *, indica que el parámetro objeto es el nombre de un objeto de un subformulario, una lista jerárquica, un list box o campo/variable imagen (en este caso, pase una cadena en objeto). Si no pasa nada en este parámetro, indica que el parámetro objeto es una tabla (tabla de formulario listado o de subformulario) o una variable (ListRef de lista jerárquica, list box o imagen) o un campo.*

El parámetro posicion permite especificar el número de la línea a mostrar o en el caso de una imagen, la coordenada vertical del pixel a mostrar.
Si no pasa el parámetro posicion, el comando provoca el desplazamiento vertical de las líneas de la lista de manera que la primera línea seleccionada en la lista sea visible. Si ninguna línea está seleccionada o si al menos una seleccionada ya es visible, el comando no hace nada.

*Si pasa este parámetro, el comando provoca el desplazamiento vertical de las líneas de la lista de manera que la línea seleccionada sea visible (seleccionada o no). Si la línea ya es visible, el comando no hace nada, excepto si se pasa el segundo parámetro * (ver a continuación).*

- *Para los formularios listados y los subformularios, este número corresponde al número de un registro en la selección actual, es decir su posición.*
- *En el caso de listas jerárquicas, el comando tiene en cuenta el estado expandido/contraído de los elementos.*
- *Para los list box, este número corresponde al número de la línea entre todas las líneas del objeto (incluyendo las líneas ocultas). Si el número pasado en posicion corresponde a una línea oculta en el listbox, el comando muestra la primera línea visible siguiente.*
  ***Nota**: recuerde que este comando se basa siempre en la representación estándar (no jerárquica) de un listbox, incluso si se muestra en modo jerárquico. Por lo tanto, el resultado puede variar dependiendo de si el listbox se muestra en modo estándar o jerárquico (ver ejemplo).*
- *Para imágenes mostradas en el formulario, posicionLinea indica el punto de coordenada vertical de la imagen a mostrar en el objeto. Pase 0 en posicionLinea para no desplazar la imagen en la dimensión vertical. El valor debe expresarse en píxeles relativos al origen de la imagen. Si el punto de coordenada vertical ya es visible en el objeto, el comando no hace nada (excepto cuando pasa el segundo parámetro * ver más adelante). La imagen debe mostrarse en el formato "Imagen truncada (no centrada)".*

El parámetro posicionH puede utilizarse en el contexto de un list box o una imagen.

- *Para los list boxes, puede pasar un número de columna en posicionH. La ejecución del comando provocará el desplazamiento horizontal del list box de manera que esta columna sea visible. Si la columna ya es visible, el comando no hace nada. Como para el desplazamiento vertical, si pasa el segundo parámetro opcional *, la columna se vuelve visible para el comando (si el list box se desplaza) se ubicará en la primera posición (ver a continuación).*
- *Para una imagen mostrada en un formulario, posicionH indica el punto de coordenada horizontal de la imagen a mostrar en el objeto. El valor debe expresarse en píxeles en relación al origen de la imagen. Si el punto de coordenada horizontal ya es visible en el objeto, el comando no hace nada (excepto cuando pasa el segundo parámetro * ver más adelante).*

*Si pasa el segundo parámetro opcional *:*

- *La línea se vuelve visible por el comando (si la lista se desplazó) se ubicará en la primera posición de la lista. Si la línea se ubica al final de la lista, esta opción no tiene efecto.*
- *En el contexto de una imagen, las coordenadas demandadas serán posicionadas en el origen de la variable imagen (0,0), incluso si las coordenadas ya son visibles en el objeto.*

***Nota:** el comando **HIGHLIGHT RECORDS** tiene un parámetro opcional * que permite delegar la gestión de desplazamiento en los formularios al comando **OBJECT SET SCROLL POSITION** .*

Ejemplo 1

Este ejemplo ilustra la diferencia de funcionamiento del comando con un list box mostrado en modo estándar y jerárquico:

```
OBJECT SET SCROLL POSITION(*;"mylistbox";4;2;*) // mostrar en la primera posición la cuarta línea y la cuarta línea y la segunda columna
```

Si esta instrucción se aplica a un list box mostrado en modo estándar:

France	Brittany	Brest	120000	...
France	Brittany	Quimper	80000	...
France	Brittany	Rennes	200000	...
France	**Normandy**	Caen	220000	...
France	Normandy	Deauville	4000	...
France	Normandy	Cherbourg	41000	...
...	...	...	...	...

... las líneas y las columnas del list box se desplazan:

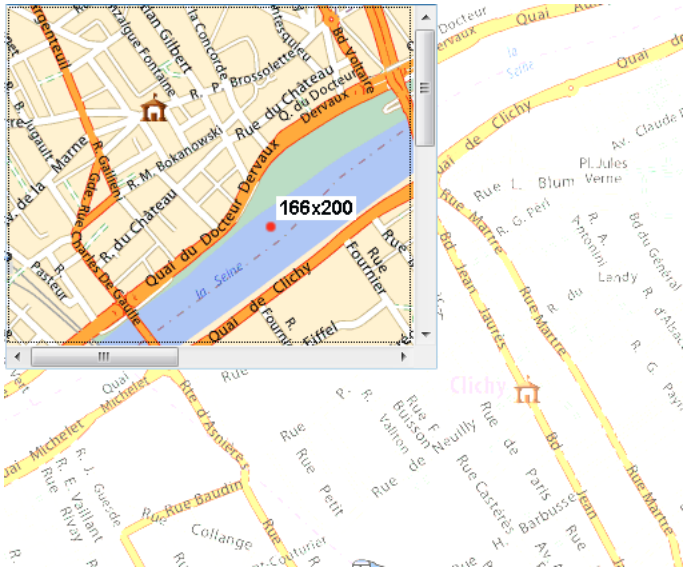
Normandy	Caen	220000	...	...
Normandy	Deauville	4000	...	...
Normandy	Cherbourg	41000	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

Por otra parte, si la misma instrucción se aplica al list box mostrado en modo jerárquico, las líneas se desplazn pero no las columnas porque la segunda columna hacer pate de la jerarquía:

V **Normandy**	
Caen	220000
Deauville	4000
Cherbourg	41000
...	

Ejemplo 2

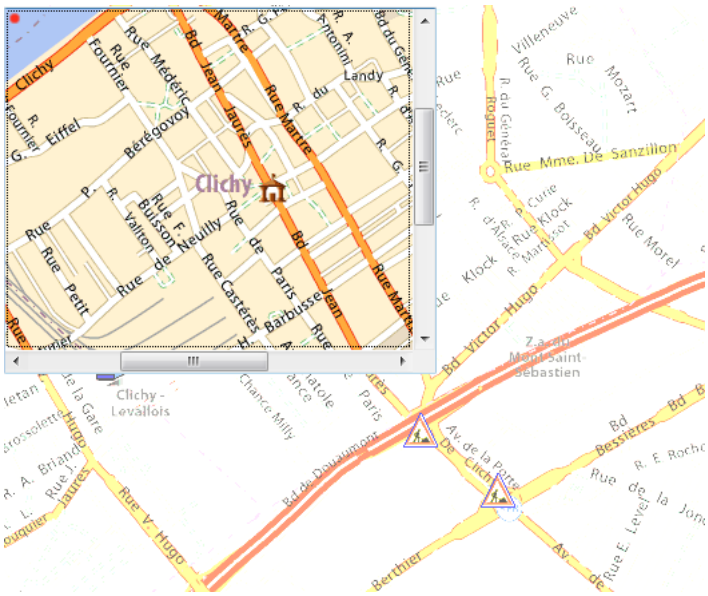
Usted quieres desplazarse por una imagen que se incluye en una variable de formulario. Este montaje muestra la parte visible de la imagen, así como el punto a mostrar (166 píxeles verticalmente y 200 píxeles horizontalmente):



Para desplazarse por la parte visible y mostrar el punto rojo al origen de la variable imagen, puede escribir:

```
OBJECT SET SCROLL POSITION(*;"myVar";166;200;*)
```

A continuación, se obtiene el siguiente resultado:



*Asegúrese de no omitir el segundo parámetro * en este caso, de lo contrario la imagen no se desplazará porque el punto definido ya se muestra.*

OBJECT SET SCROLLBAR

OBJECT SET SCROLLBAR ({* ;} objeto ; horizontal ; vertical)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Variable (si se omite *)
horizontal	Booleano, Entero largo	➡	True = mostrar, False = ocultar
vertical	Booleano, Entero largo	➡	True = mostrar, False = ocultar

Descripción

El comando ***OBJECT SET SCROLLBAR*** *le permite mostrar u ocultar las barras de desplazamiento horizontal o vertical en el objeto designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa el parámetro opcional *, indica que el parámetro objeto es una variable. En este caso, no se pasa una cadena sino una referencia de una variable. Para mayor información sobre nombres de objetos, consulte la sección* ***Propiedades de los objetos****.*
Pase en los parámetros horizontal y vertical los valores indicando si las barras de desplazamiento correspondientes deben ser mostrarse. Puede pasar valores booleanos (True=mostrado, False=oculto), o valores numéricos (0=oculto, 1=mostrado, 2=modo automático). El uso de valores numéricos le da acceso al modo automático, donde sólo se muestran las barras de desplazamiento cuando sea necesario.

.

Objetos con barras de desplazamiento	Ocultar barra de desplazamiento	Mostrar barra de desplazamiento	Modo automático
Text object fields and variables	False o 0	True o 1	no disponible
Campos y variable objeto texto	False o 0	True o 1	2
List boxes	False o 0	True o 1	2
Listas jerárquicas	False o 0	True o 1	2
Subformularios	False o 0	True o 1	no disponible

Por defecto, se muestran las barras de desplazamiento.

Nota: *para obtener más información sobre el modo automático, consulte* ***Barras de desplazamiento****.*

OBJECT SET SHORTCUT

OBJECT SET SHORTCUT ({* ;} objeto ; tecla {; modificadores})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable o campo(si se omite *)
tecla	Cadena	➡	Tecla a asociar al objeto
modificadores	Entero largo	➡	Máscara o combinación de máscaras de teclas de modificación

Descripción

El comando ***OBJECT SET SHORTCUT*** *permite definir o modificar dinámicamente el atajo de teclado asociado al objeto o a los objetos designados por los parámetros objeto y * para el proceso actual.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, indica que el parámetro objeto es una variable o un campo. En este caso, pase una referencia en lugar de un nombre.*

En el parámetro tecla, pase una cadena indicando la tecla a asociar al objeto. Puede pasar:

- *un nombre de tecla estándar, por ejemplo "B"*
- *o una constante del tema* ***Atajos de teclado*** *(o su valor) :*

Constante	Tipo	Valor
Shortcut with Backspace	Cadena	[backspace]
Shortcut with Carriage Return	Cadena	[return]
Shortcut with Delete	Cadena	[del]
Shortcut with Down arrow	Cadena	[down arrow]
Shortcut with End	Cadena	[end]
Shortcut with Enter	Cadena	[enter]
Shortcut with Escape	Cadena	[esc]
Shortcut with F1	Cadena	[F1]
Shortcut with F10	Cadena	[F10]
Shortcut with F11	Cadena	[F11]
Shortcut with F12	Cadena	[F12]
Shortcut with F13	Cadena	[F13]
Shortcut with F14	Cadena	[F14]
Shortcut with F15	Cadena	[F15]
Shortcut with F2	Cadena	[F2]
Shortcut with F3	Cadena	[F3]
Shortcut with F4	Cadena	[F4]
Shortcut with F5	Cadena	[F5]
Shortcut with F6	Cadena	[F6]
Shortcut with F7	Cadena	[F7]
Shortcut with F8	Cadena	[F8]
Shortcut with F9	Cadena	[F9]
Shortcut with Help	Cadena	[help]
Shortcut with Home	Cadena	[home]
Shortcut with Left arrow	Cadena	[left arrow]
Shortcut with Page down	Cadena	[page down]
Shortcut with Page up	Cadena	[page up]
Shortcut with Right arrow	Cadena	[right arrow]
Shortcut with Tabulation	Cadena	[tab]
Shortcut with Up arrow	Cadena	[up arrow]

En el parámetro modificadores, pase una o más teclas de modificación a asociar al atajo de teclado. Para definir el parámetro modificadores, pase una o más de las siguientes constantes de tipo "Mask" del tema ***Eventos (Modificadores)****:*

Constante	Tipo	Valor	Comentario
Command key mask	Entero largo	256	Tecla Ctrl en Windows, Tecla Comando en OS X
Control key mask	Entero largo	4096	Tecla Ctrl bajo OS X, o clic derecho en Windows y OS X
Option key mask	Entero largo	2048	Tecla Alt (también llamada Opción en OS X)
Shift key mask	Entero largo	512	Windows y OS X

Nota*: si omite el parámetro modificadores, el objeto se activa tan pronto como se presiona la tecla definida. Por ejemplo, si se asocia la tecla "H" a un botón, este botón se activa cada vez que presione la tecla H. Este funcionamiento ser reserva para interfaces específicas.*

Ejemplo

Usted quiere asociar un atajo de teclado diferente en función del lenguaje actual de la aplicación. En el evento On Load form, puede escribir:

```
Case of
    vLang="FR"
    OBJECT SET SHORTCUT(*;"SortButton";"T";Command key mask+Shift key mask) // Ctrl+Mayús+T en francés
    vLang="US"
    OBJECT SET SHORTCUT(*;"SortButton";"O";Command key mask+Shift key mask) // Ctrl+Mayús+O en inglés
End case
```

OBJECT SET STYLE SHEET

OBJECT SET STYLE SHEET ({* ;} objeto ; nomHojaEstilo)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
nomHojaEstilo	Texto	➡	Nombre de la hoja de estilo

Descripción

El comando ***OBJECT SET STYLE SHEET*** *modifica, para el proceso actual, la hoja de estilo asociada al objeto(s) designado(s) por los parámetros objeto y * . Una hoja de estilo modifica la fuente, el tamaño de fuente y (excepto para las hojas de estilo automáticas) el estilo de fuente.*

*Al pasar el parámetro opcional * se indica que el parámetro objeto es un nombre de objeto (cadena). Si no se pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

Pase en el parámetro nomHojaEstilo, el nombre de la hoja de estilo a aplicar al objeto. También puede pasar:

- *el nombre de una hoja de estilos existente (si la hoja de estilo no existe, se devuelve un error, que puede interceptar utilizando un método instalado por el comando* ***ON ERR CALL****).*
- *una cadena vacía ("") para no aplicar la hoja de estilo al objeto.*
- *una de las siguiente constantes del tema "****Estilos de fuente****" para aplicar una hoja de estilo automática:*

Constante	Tipo	Valor	Comentario
Automatic style sheet	Cadena	__automatic__	Se utiliza de forma predeterminada para todos los objetos
Automatic style sheet_additional	Cadena	__automatic_additional_text__	Soportado por los textos estáticos, campos y variables solamente. Se utiliza para texto adicional en las cajas de diálogo.
Automatic style sheet_main	Cadena	__automatic_main_text__	Soportado por los textos estáticos, campos y variables solamente. Se utiliza para texto principal en las cajas de diálogo.

Si una hoja de estilo ya se había asociado al objeto en modo Diseño, la llamada de este comando la reemplaza para el proceso actual.

Si durante la sesión, utiliza los comandos ***ST SET ATTRIBUTES****,* ***ST SET TEXT*** *o* ***OBJECT SET FONT*** *en el objeto con el fin de modificar su fuente o el tamaño de fuente, la referencia a la hoja de estilos se borra automáticamente del objeto, incluso si asigna los mismos atributos que los de la hoja de estilos. Sin embargo, si se modifica el estilo (negrita, cursiva, etc.), por ejemplo con los comandos* ***ST SET ATTRIBUTES*** *o* ***OBJECT SET FONT STYLE****, se añaden estas nuevas propiedades a la hoja de estilo por la duración de la sesión.*

OBJECT SET SUBFORM

OBJECT SET SUBFORM ({* ;} objeto {; aTabla}; subFormDet {; subFormList})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
aTabla	Tabla	➡	Tabla de formulario (si tabla de formulario)
subFormDet	Texto, Objeto	➡	Nombre del formulario detallado de subformulario
subFormList	Texto, Objeto	➡	Nombre del formulario listado de subformulario (formulario tabla)

Descripción

*El comando **OBJECT SET SUBFORM** permite modificar dinámicamente el formulario detallado así como también, opcionalmente, el formulario listado asociado al objeto subformulario designado por los parámetros objeto y *.*

***Nota**: este comando no permite cambiar el tipo de subformulario mismo (lista o página). Esta propiedad sólo se puede configurar en modo Diseño.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

En el parámetro aTabla, pase la tabla de los formularios a utilizar. Este parámetro es opcional; puede omitirlo si especifica un formulario proyecto como subformulario detallado.

En los parámetros subFormDet y subFormList, pase:

- *el nombre de un formulario, o*
- *la ruta* (en sintaxis POSIX) a un archivo .json válido que contiene una descripción del formulario a usar (ver **Ruta de archivo del formulario**), o*
- *un objeto que contiene una descripción del formulario.*

**A diferencia de otros comandos relacionados con formularios, las rutas de archivo de OBJECT SET SUBFORM son relativas al formulario principal del subformulario.*

***Nota**: el parámetro subFormList sólo se puede pasar cuando modifica un subformulario de tipo lista.*

Cuando modifica un subformulario página, el comando puede ejecutarse en cualquier momento; las selecciones actuales no se modifican. Sin embargo, cuando modifica un subformulario listado, sólo puede modificarse cuando muestra la lista. Si el comando se ejecuta cuando el formulario detallado se muestra después de un doble clic en la lista, se genera un error.

OBJECT SET TEXT ORIENTATION

OBJECT SET TEXT ORIENTATION ({* ;} objeto ; orientacion)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
orientacion	Entero largo	➡	Valor de orientación del objeto

Descripción

*El comando **OBJECT SET TEXT ORIENTATION** modifica la orientación del contenido del objeto o de los objetos designados por los parámetros objeto y * para el proceso actual.*

*La propiedad "Orientación", disponible en el editor de formularios, realiza rotaciones de áreas de texto de manera permanente en sus formularios. A diferencia de esta propiedad, el comando **OBJECT SET TEXT ORIENTATION** aplica la rotación al contenido del objeto, pero no al objeto en sí. Para obtener más información, consulte el Manual de Diseño.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*
Sólo los textos estáticos, así como las variables y campos no editables se pueden girar. Si se aplica este comando a un objeto que no sea compatible con la orientación de texto, el comando no hace nada.

*En el parámetro orientacion, pase la orientación absoluta que desea asignar al objeto. Debe utilizar una de las siguientes constantes, del tema "**Propiedades de los objetos**":*

Constante	Tipo	Valor	Comentario
Orientation 0°	Entero largo	0	Sin rotación (valor por defecto)
Orientation 180°	Entero largo	180	Orientación del texto a 180° en el sentido horario
Orientation 90° left	Entero largo	270	Orientación del texto a 90° en el sentido antihorario
Orientation 90° right	Entero largo	90	Orientación del texto a 90° en el sentido horario

***Nota**: sólo se admiten los ángulos correspondientes a estos valores. Si pasa cualquier otro valor, se ignorará.*

Ejemplo

Desea aplicar una orientación de 270° a una variable en su formulario:

```
OBJECT SET ENTERABLE(*;"myVar";False)
 // Obligatorio si la variable es editable
OBJECT SET TEXT ORIENTATION(*;"myVar";Orientation 90° left)
```

OBJECT SET THREE STATES CHECKBOX

OBJECT SET THREE STATES CHECKBOX ({* ;} objeto ; tresEst)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
tresEst	Booleano	➡	True = casilla de selección de tres estados, False = casilla de selección estándar

Descripción

El comando ***OBJECT SET THREE STATES CHECKBOX*** *modifica para el proceso actual, la propiedad de "Tres Estados" de la(s) casilla(s) de selección designada(s) por los parámetros objeto y * .*

La opción de "Tres estados" permite utilizar el estado adicional "semi seleccionado" para las casillas a seleccionar. Para obtener más información, consulte ***Casillas de selección de tres estados*** *en el manual de Diseño.*

*Al pasar el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

Este comando sólo se aplica a casillas de selección asociadas a las variables, y no a los campos booleanos representados como casillas de selección.

En el parámetro tresEst, pase ***True*** *para activar el modo "tres estados", o* ***False*** *para desactivarlo.*

OBJECT SET TITLE

OBJECT SET TITLE ({* ;} objeto ; titulo)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Variable (si se omite *)
titulo	Cadena	➡	Nuevo título para el objeto

Descripción

El comando OBJECT SET TITLE cambia el título de los objetos especificados por objeto y lo reemplaza por el valor pasado en titulo.

*Si especifica el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena) . Si omite el parámetro opcional *, indica que el parámetro objeto es una variable. En este caso, usted especifica una referencia de un campo o de una variable (variable tipo objeto únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección* ***Propiedades de los objetos****.*

OBJECT SET TITLE aplica a todos los tipos de objetos simples que contienen una etiqueta:

- *botones y botones 3D,*
- *casillas de selección y casillas de selección 3D,*
- *botones radio y botones radio 3D,*
- *encabezados de listbox,*
- *textos estáticos,*
- *áreas de grupo.*

Generalmente, este comando se aplica a un objeto a la vez. El área de título del objeto debe ser lo suficientemente grande para acomodar el texto; si no lo es, el texto se trunca.

No utilice retornos de carro en titulo.

Ejemplo 1

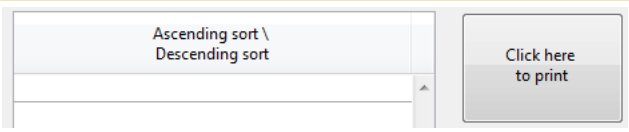
El siguiente ejemplo es el método de objeto de un botón de búsqueda ubicado en el área de pie de página de un formulario de salida mostrado por el comando ***MODIFY SELECTION****. El método busca una tabla; dependiendo de los resultados de búsqueda, activa o desactiva un botón titulado bEliminar y cambia su título:*

```
QUERY([Personas];[Personas]Nombre=vNombre)
Case of
   :(Records in selection([Personas])=0)  // No se encontró ninguna persona
      OBJECT SET TITLE(bDelete;" Borrar")
      OBJECT SET ENABLED(bDelete;False)
   :(Records in selection([Personas])=1)  // Se encontró una persona
      OBJECT SET TITLE(bDelete;"Borrar persona")
      OBJECT SET ENABLED(bDelete;True)
   :(Records in selection([Personas])>1)  // Se encontraron varias personas
      OBJECT SET TITLE(bDelete;"Borrar personas")
      OBJECT SET ENABLED(bDelete;True)
End case
```

Ejemplo 2

Usted quiere insertar los títulos en dos líneas:

```
OBJECT SET TITLE(*;"header1";"Ascending sort \\\ \\Descending sort")
OBJECT SET TITLE(*;"button1";"Click here \\to print")
```



OBJECT SET VERTICAL ALIGNMENT

OBJECT SET VERTICAL ALIGNMENT ({* ;} objeto ; alineacion)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Variable (si se omite *)
alineacion	Entero largo	➡	Código de alineación

Descripción

El comando ***OBJECT SET VERTICAL ALIGNMENT*** *modifica por programación el tipo de alineación vertical aplicada al objeto designado por los parámetros objeto y *.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Si no pasa este parámetro, esto indica que el parámetro objeto es una variable. En este caso, se pasa una referencia de variable en lugar de una cadena.*

En alineacion, puede pasar una de las siguientes constantes, del tema ***Propiedades de los objetos****:*

Constante	Tipo	Valor
Align bottom	Entero largo	4
Align center	Entero largo	3
Align default	Entero largo	1
Align top	Entero largo	2

La alineación vertical puede aplicarse a los siguientes tipos de objetos de formulario:

- *list boxes,*
- *columnas de list box,*
- *encabezados y pies de list box.*

OBJECT SET VISIBLE

OBJECT SET VISIBLE ({* ;} objeto ; visible)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Campo o variable (si se omite *)
visible	Booleano	➡	True para visible, False para invisible

Descripción

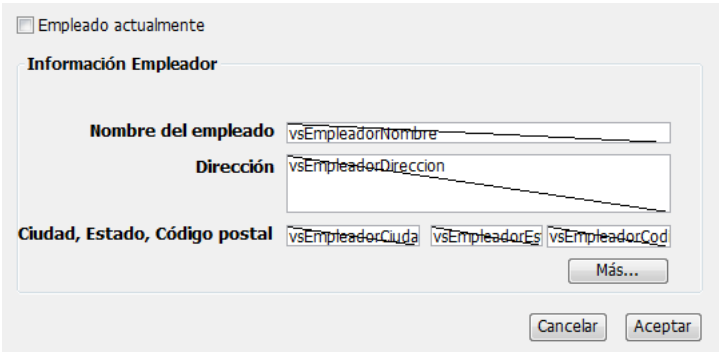
*El comando **OBJECT SET VISIBLE** muestra u oculta los objetos especificados por objeto.*

*Si especifica el parámetro opcional *, indica que el parámetro objeto designa el nombre de un objeto (una cadena). Si no pasa el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, usted especifica una referencia de un campo o de una variable (campo o variable tipo objeto únicamente) en lugar de una cadena. Para mayor información sobre nombres de objetos, consulte la sección .*

*Si pasa visible igual a **TRUE**, los objetos son mostrados. Si pasa visible igual a **FALSE**, los objetos se ocultan.*

Ejemplo

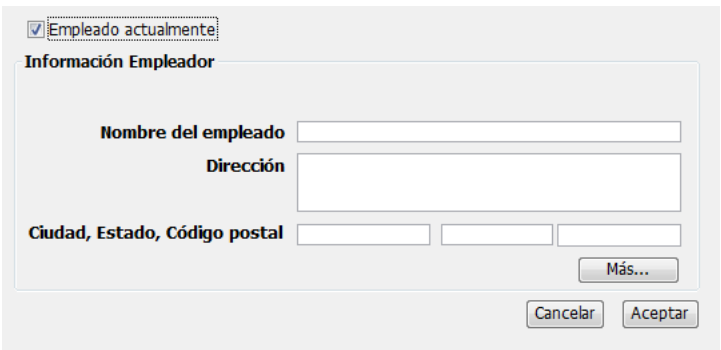
Este es un formulario típico en el entorno Diseño:



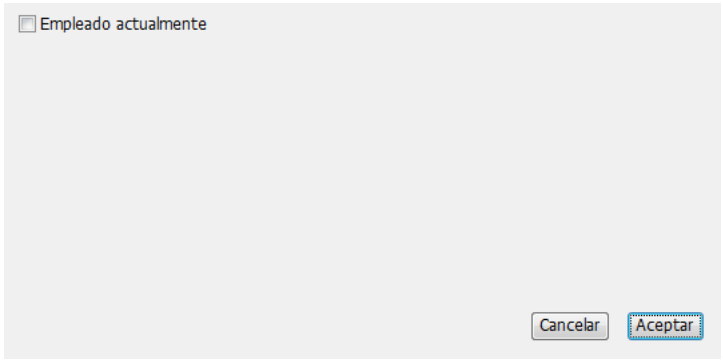
*Los objetos en el área de grupo **Información del empleador** cada uno tiene un nombre de objeto que contiene la expresión "empleador" (incluyendo el área de grupo). Cuando la casilla de selección **Empleado actualmente** está seleccionada, los objetos deben ser visibles; cuando la casilla no está seleccionada, los objetos deben ser invisibles.*
Este es el método de objeto de la casilla de selección:

```
   ` Método de objeto de Casilla de selección cbEmpleadoActualmente
Case of
   :(Form event=On Load)
      cbEmpleadoActualmente:=1

   :(Form event=On Clicked)
   ` Ocultar o mostrar todos los objetos cuyo nombre contiene "emp"
      OBJECT SET VISIBLE(*;"@emp@";cbEmpleadoActualmente # 0)
   ` Pero siempre conservar la casilla de selección visible
      OBJECT SET VISIBLE(cbEmpleadoActualmente;True)
End case
```

Por lo tanto, en ejecución, el formulario se ve así:



o:


Empleado actualmente
Cancelar
Aceptar

_o_DISABLE BUTTON

_o_DISABLE BUTTON ({* ;} objeto)

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	⇒	Nombre de objeto (si se especifica *), o Variable (si se omite *)

Nota de compatibilidad

*El comando **_o_DISABLE BUTTON** fue declarado obsoleto en 4D desde la versión 12 y se conserva únicamente por razones de compatibilidad. Su alcance, incluye todas las instancias de la variable designada y no sólo las del formulario actual, no corresponde a las del tema "Objetos (Formularios)".*
***_o_DISABLE BUTTON** puede reemplazarse favorablemente con el comando **OBJECT SET ENABLED**.*

_o_ENABLE BUTTON

_o_ENABLE BUTTON ({* ;} objeto)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *), o Variable (si se omite *)

Nota de compatibilidad

A partir de la versión 12 el comando ***_o_ENABLE BUTTON*** *es declarado obsoleto en 4D por razones de compatibilidad. Su alcance global, incluye todas las instancias de la variable designada y no sólo las del formulario actual, no corresponde a las de los comandos del tema "Objetos (Formularios)".*
_o_ENABLE BUTTON *y* ***_o_DISABLE BUTTON*** *puede reemplazarse favorablemente por los comandos* ***OBJECT SET ENABLED*** *y* ***OBJECT Get enabled****.*

_o_OBJECT Get action

_o_OBJECT Get action ({* ;} objeto) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si * se especifica) o Campo o variable (si * se omite)
Resultado	Texto	⮌	Acción estándar asociada

Nota de compatibilidad

*Este comando es obsoleto a partir de 4D v16 R3. Se sustituye por el comando **OBJECT Get action**, actualizado que se beneficia del conjunto de nombres de acción estándar extendidos basados en texto.*

Descripción

*El comando **_o_OBJECT Get action** devuelve devuelve el código de la acción estándar asociada al objeto designado por los parámetros objeto y *.*

***Nota:** las constantes de acción basadas en código son obsoletas a partir de 4D v16 R3. Para más información, consulte la descripción del comando **OBJECT Get action** (actualizado).*

Objetos (Lenguaje)

- *Estructura de los objetos de lenguaje 4D*
- *Uso de la notación objeto*
- *Objetos y colecciones compartidos*
- *New object*
- *New shared object*
- *OB Copy*
- *OB Get*
- *OB GET ARRAY*
- *OB GET PROPERTY NAMES*
- *OB Get type*
- *OB Is defined*
- *OB Is empty*
- *OB REMOVE*
- *OB SET*
- *OB SET ARRAY*
- *OB SET NULL*
- *Storage*

Estructura de los objetos de lenguaje 4D

---

*Los comandos del tema **Objetos (Lenguaje)** permiten crear y trabajar con los datos en forma de objeto. Esta funcionalidad amplía las oportunidades de intercambio entre 4D y todo tipo de aplicación que soporte los datos estructurados.*

Todos los comandos de este tema tienen en cuenta los objetos 4D siguientes:

- *Las variables objeto o arrays objeto creados e inicializados utilizando los comandos (tema "**Compilador**") o **ARRAY OBJECT** (tema "**Arrays**").*
- *Los campos objeto desde la base de datos 4D (ver **Tipos de campos 4D**).*

*Para más información sobre la estructura de objetos 4D "nativos", consulte el párrafo **Objeto** en la sección **Tipos de datos**.*

Uso de la notación objeto

Presentación

*Puede gestionar objetos del lenguaje 4D utilizando la **notación objeto** para obtener o definir sus valores. Por motivos de compatibilidad, esta funcionalidad requiere que active explícitamente una opción de compatibilidad. Una vez activada la notación de objetos, puede utilizarla en todas partes en 4D donde se esperen expresiones.*

¿Dónde utilizar la notación objeto?

Cada valor de propiedad accedido a través de la notación objeto se considera una expresión. Cuando la notación objeto está habilitada en su base de datos (ver abajo), puede usar estos valores dondequiera que se esperen expresiones 4D:

- *En el código 4D, ya sea escrito en los métodos (editor de métodos) o externalizado (fórmulas, archivos de etiquetas 4D procesados por **PROCESS 4D TAGS** o el servidor Web, archivos de exportación, documentos 4D Write Pro, etc.)*
- *En las áreas Expresión del depurador y del explorador de ejecución,*
- *En la lista de propiedades del editor de formularios para los objetos de formularios: campo Variable o Expresión, así como también varias expresiones utilizables en los list box de tipo selección y sus columnas (fuentes de datos, color de fondo, estilo o color de fuente).*

Inicialización

*Los objetos manipulados mediante la notación de objetos deben haberse inicializado utilizando el comando **New object**, de lo contrario, el acceso a sus propiedades de lectura y escritura generará un error de sintaxis.*

Ejemplo:

```
C_OBJECT($obVar) //creación de una variable 4D de tipo objeto
$obVar:=New object //inicialización del objeto y asignación a la variable 4D
```

*El mismo principio aplica a campos de tipo **Objeto**:*

```
CREATE RECORD([Person]) //Agregar un nuevo registro a una tabla que contiene un campo objeto
[Person]Data_o:=New object //Inicialización del objeto y asignación al campo 4D
```

Generalidades

La notación de objetos se puede utilizar para acceder a valores de propiedad de objeto y a elementos de colección a través de una cadena de tokens.

Propiedades de los objetos

Con la notación objeto, las propiedades de los objeto se pueden acceder de dos maneras:

- *Utilizando un símbolo "punto":*

```
object.propertyName
```

Ejemplo:

```
employee.name:="Smith"
```

- *Utilizando una cadena entre corchetes:*

```
object["propertyName"]
```

Ejemplo:

```
$vName:=employee["name"]
```

Como el valor de una propiedad de objeto puede ser un objeto o una colección, la notación objeto requiere una secuencia de símbolos para acceder a las sub-propiedades, por ejemplo:

```
$vAge:=employee.children[2].age
```

La notación objeto está disponible con todo elemento de lenguaje que pueda contener o devolver un objeto, es decir:

- *con los **objetos** mismo (almacenados en variables, campos, propiedades de objetos, arrays de objetos o elementos de colecciones).*

Ejemplos:

```
$age:=$myObjVar.employee.age  //variable
$addr:=[Emp]data_obj.address  //campo
$city:=$addr.city  //propiedad de un objeto
$pop:=$aObjCountries{2}.population  //array de objetos
$val:=$myCollection[3].subvalue  //elemento de colección
```

- *con los **comandos 4D** que devuelven objetos.*
  Ejemplo:

```
$measures:=Get database measures.DB.tables
```

- *con los **métodos proyecto** que devuelven objetos.*
  Ejemplo:

```
 // MyMethod1
C_OBJECT($0)
$0:=New object("a";10;"b";20)

 //myMethod2
$result:=MyMethod1.a  //10
```

- *con las **Colecciones** (sólo la propiedad longitud, ver párrafo siguiente).*
  Ejemplo:

```
myColl.length  //tamaño de la colección
maxSal:=myColl.max("salary")
```

Elementos de colecciones y propiedad length

Para acceder a un elemento de colección, debe pasar el número del elemento al interior de los corchetes:

```
collectionName[expression]
```

***Nota:** para mayor información sobre las variables de tipo colección, consulte la sección **Colección**.*
Puede pasar toda expresión 4D válida que devuelva un entero positivo en expresión. Ejemplos:

```
myCollection[5]  //acceso al sexto elemento de la colección
myCollection[$var]
```

***Nota**: tenga en cuenta que los elementos de colección están numerados desde 0.*
Puede asignar un valor a un elemento de colección mediante la notación objeto:

```
myCol[10]:="My new element"
```

*Si este índice de elemento está más allá del último elemento existente de la colección, la colección se redimensiona automáticamente y todos los elementos intermediarios nuevos obtienen el valor **null**:*

```
C_COLLECTION(myCol)
myCol:=New collection("A";"B")
myCol[5]:="Z"
 //myCol[2]=null
 //myCol[3]=null
 //myCol[4]=null
```

Propiedad length

*La propiedad **length** está disponible automáticamente para todas las colecciones y devuelve el tamaño de la colección, es decir, el número de elementos que contiene. Puede acceder a esta propiedad de dos maneras:*

- *Utilizando un símbolo "punto", por ejemplo:*

```
$vSize:=myCollection.length
```

- *Usando una cadena entre corchetes, por ejemplo:*

```
$vSize:=myCollection["length"]
```

*Tenga en cuenta que la propiedad **length** sólo se puede leer, no se puede modificar.*

Punteros

Los valores de propiedades se pueden acceder a través de punteros. El uso de la notación de objetos con punteros es muy similar al uso de la notación de objetos directamente con los objetos, excepto que el símbolo "punto" debe omitirse.

- *Acceso directo:*

```
pointerOnObject->propertyName
```

- *Acceso por nombre:*

```
pointerOnObject->["propertyName"]
```

Ejemplo:

```
C_OBJECT(vObj)
C_POINTER(vPtr)
vObj:=New object
vObj.a:=10
vPtr:=->vObj
x:=vPtr->a //x=10
```

Valor Null

*Cuando se utiliza la notación de objetos, el valor **null** se soporta a través del comando **Null**. Este comando se puede utilizar para asignar o comparar un valor **null** a las propiedades de los objetos o a los elementos de colecciones, por ejemplo:*

```
myObject.address.zip:=Null
Si(myColl[2]=Null)
```

*Para más información, consulte la descripción del comando **Null**.*

Valor indefinido

La evaluación de una propiedad de objeto a veces puede producir un valor indefinido. Este es el caso, por ejemplo, cuando se escribe:

```
C_OBJECT($o)
$o:=New object("a";2)
$val:=$o.b //$val es indefinido
```

Normalmente, al intentar leer o asignar expresiones indefinidas, 4D generará errores. Esto no sucede en los siguientes casos:

- *La lectura de una propiedad de un objeto o valor indefinido devuelve indefinido; asignar un valor indefinido a variables (excepto arrays) tiene el mismo efecto que llamar a **CLEAR VARIABLE** con ellos:*

```
C_OBJECT($o)
C_LONGINT($val)
$val:=10 //$val=10
$val:=$o.a //$o.a es indefinido (sin error), y asignar este valor borra la variable
 //$val=0
```

- *Leer la propiedad **length** de una colección indefinida produce 0:*

```
C_COLLECTION($c) //variable creada pero no se define ninguna colección
$size:=$c.length //$size = 0
```

- *Un valor indefinido pasado como parámetro a un método de proyecto se convierte automáticamente a 0 o "" según el tipo de parámetro declarado.*

```
C_OBJECT($o)
mymethod($o.a) //pasa un parámetro no definido

 //En método mymethod
C_TEXT($1) //tipo de parámetro es texto
 // $1 contiene ""
```

- *Una expresión de condición se convierte automáticamente a false cuando se evalúa como indefinida con las palabras claves **If** y **Case of**:*

```
C_OBJECT($o)
If($o.a) // false
End if
Case of
   :($o.a) // false
End case
```

- *Asignar un valor indefinido a una propiedad de objeto existente reinicializa o borra su valor, dependiendo de su tipo:*
  - *Objeto, colección, puntero: Null*
  - *Imagen: Imagen vacía*
  - *Booleano: False*
  - *Cadena: ""*
  - *Número: 0*
  - *Fecha: !00-00-00! si está activada la configuración "Usar tipo de fecha en lugar de formato de fecha ISO en objetos", de lo contrario ""*
  - *Hora: 0 (número de ms)*
  - *Indefinido, Null: sin cambio*

```
C_OBJECT($o)
$o:=New object("a";2)
$o.a:=$o.b //$o.a=0
```

- *Asignar un valor indefinido a una propiedad de objeto no existente no hace nada.*

Cuando se esperan expresiones de un tipo dado en su código 4D, puede asegurarse de que tienen el tipo correcto incluso cuando se evalúan como indefinido rodeándolas con el comando cast 4D apropiado: ***String***, ***Num***, ***Time***, ***Date***, ***Bool****. Estos comandos devuelven un valor vacío del tipo especificado cuando la expresión se evalúa como indefinida. Por ejemplo:*

```
$myString:=Lowercase(String($o.a.b)) //asegúrese de obtener un valor de cadena incluso si no está definido
 //para evitar errores en el código
```

Identificadores de propiedades de objetos

Las reglas de nombres de miembros tokens (nombres de propiedades de objetos que se acceden utilizando notación de objetos) son más restrictivas que las que se aplican a los nombres de identificación 4D estándar. Deben cumplir con la gramática del identificador de JavaScript (ver ECMA Script standard):

- *El primer carácter debe ser una letra, un guion bajo (_), o un signo de dólar ($),*
- *Los siguientes caracteres pueden ser cualquier letra o dígito o un guion bajo o signo de dólar (los caracteres de espacio NO están permitidos),*
- *Son sensibles a mayúsculas y minúsculas.*

Notas:

- *No se permite usar un campo tabla como índice de colección, por ejemplo a.b [[Table1]Id]. Debe utilizar una variable intermedia.*
- *La creación de atributos de objeto mediante una cadena entre corchetes permite anular las reglas de ECMA Script. Por ejemplo, el atributo $o["My Att"] es válido en 4D, a pesar del espacio. En este caso, sin embargo, no será posible utilizar la notación de puntos con este atributo.*

Activación de la notación objeto

En todas las versiones, 4D siempre ha aceptado puntos (.) y corchetes ([y]) en los nombres tokenizados de los objetos de la base de datos (tablas, campos, variables y métodos).

Sin embargo, estos caracteres se utilizan para identificar tokens de lenguaje en la notación de objetos estándar. Por lo tanto, las bases de datos que utilizan nombres que contienen puntos o corchetes no son compatibles con la notación de objetos estándar ya que interpretaciones erróneas podrían romper el código existente. Por ejemplo, si se escribe el siguiente código:

a.b
a.b:=c[1]

...4D no podía saber si a.b y c[1] representan nombres de variables estándar o si b es una propiedad del objeto a y c el segundo elemento de una colección de objetos c.

Por consiguiente:

- *En bases convertidas de versiones anteriores a 4D v17, debe seleccionar una opción específica (ver abajo) que indique que desea utilizar la notación de objetos. Al seleccionar esta opción, usted declara que su código está "listo para la notación de objetos", es decir, no utiliza ningún nombre que contenga "." o "[]".*
- *Una funcionalidad específica del CSM le ayuda a detectar los nombres que son incompatibles con la notación de objetos. Es muy recomendable utilizar esta función antes de activar la opción (consulte la sección* ***Página Verificar*** *del capítulo "CSM"). Como es habitual, se recomienda trabajar con una copia del archivo de estructura.*
- *A partir de 4D v17 (v16 R4), los caracteres "." y "[]" ya no se permiten en los nombres de objetos tokenizados.*

Configuración de compatibilidad

Para poder utilizar la notación de objetos en bases creadas en versiones anteriores a 4D v17, debe seleccionar la opción ***Utilizar la notación de objetos para acceder a las propiedades de los objetos (Se requiere Unicode).*** *en la página* ***Compatibilidad*** *del diálogo Configuración de la base de datos:*

Esta configuración modificará el estado interno del archivo de estructura y no se puede deshacer. Para más información, consulte ***Página Compatibilidad****.*

Nota: *los componentes pueden tener una configuración diferente de la base local.*

Ejemplos

Utilizar notación de objetos simplifica el código 4D mientras se manipulan objetos. Tenga en cuenta, sin embargo, que la notación basada en comandos sigue siendo totalmente soportada.

- *Escribir y leer objetos (este ejemplo compara la notación de objetos y la notación de comandos):*

```
 // Uso de la notación de objetos
C_OBJECT($myObj) //declara un objeto variable 4D
$myObj:=New object //crea un objeto y lo asigna a la variable
$myObj.age:=56
$age:=$myObj.age //56

 // Usando la notación de comandos
C_OBJECT($myObj2) //declara un objeto variable 4D
OB SET($myObj2;"age";42) //crea un objeto y agrega la propiedad age
$age:=OB Get($myObj2;"age") //42

 // Por su puesto, ambas notaciones se pueden mezclar
C_OBJECT($myObj3)
OB SET($myObj3;"age";10)</p><ul><li><span id="result_box"lang="es"><span title="Create a property and assign values, including objects: ">Crea una propiedad y asigna valores, incluidos los objetos:
</span></span></li></ul><p>[#code4D]C_OBJECT($Emp)
$Emp:=New object
$Emp.city:="London" //crea la propiedad city y establece su valor en "London"
$Emp.city:="Paris" //modifica la propiedad city
$Emp.phone:=New object("office";"123456789";"home";"0011223344")
 //crea la propiedad phone y define su valor en un objeto
```

- *Obtiene un valor en un sub objeto es muy sencillo utilizando la notación de objeto:*

```
$vCity:=$Emp.city //"Paris"
$vPhone:=$Emp.phone.home //"0011223344"
```

- *Puede acceder a las propiedades como cadenas usando el operador [ ]*

```
$Emp["city"]:="Berlin" //modifica la propiedad city
 //esto puede ser útil para crear las propiedades con la ayuda de variables
C_TEXT($addr)
$addr:="address"
For($i;1;4)
   $Emp[$addr+String($i)]:=""
End for
 // crea 4 propiedades vacías "address1...address4" en el objeto $Emp
```

Objetos y colecciones compartidos

Definición

*Los **objetos compartidos** y las **colecciones compartidas** son objetos específicos y colecciones cuyos contenidos se comparten entre procesos. A diferencia de **Variables interproceso**, los objetos compartidos y las colecciones compartidas tienen la ventaja de ser compatibles con **Procesos 4D apropiativos**: se pueden pasar por referencia como parámetros a comandos como **New process** o **CALL WORKER**.*

*Los objetos compartidos y las colecciones compartidas se pueden almacenar en las variables declaradas con los comandos estándar **C_OBJECT** y **C_COLLECTION**, pero se deben crear instancias utilizando comandos específicos:*

- *para crear un objeto compartido, use el comando **New shared object**,*
- *para crear una colección compartida, use el comando **New shared collection**.*

***Nota:** los objetos compartidos y las colecciones compartidas se pueden definir como propiedades de objetos/elementos de colecciones estándar (no compartidos).*

*Para modificar un objeto/colección compartido, debe llamarse la estructura **Use...End use**. Leer un valor de objeto/colección compartido no requiere **Use...End use**.*

*Un único catálogo global devuelto por el comando **Storage** siempre está disponible en toda la base y sus componentes y se puede usar para almacenar todos los objetos y colecciones compartidos.*

Utilizar objetos o colecciones compartidos

*Una vez instanciado con los comandos **New shared object** o **New shared collection**, los atributos objeto/colección compartidos y elementos se pueden modificar o leer desde cualquier proceso.*

Modificación

Las modificaciones se pueden aplicar a objetos compartidos y colecciones compartidas:

- *agregar o eliminar propiedades de objetos,*
- *agregar o editar valores (siempre que sean soportados en objetos compartidos), incluyendo otros objetos o colecciones compartidos (que crea un grupo compartido, ver más adelante).*

*Sin embargo, todas las instrucciones de modificación en un objeto compartido o colección deben estar rodeadas por las palabras claves **Use...End use**, de lo contrario, se generará un error.*

```
$s_obj:=New shared object("prop1";"alpha")
Use($s_obj)
   $s_obj.prop1:="omega"
End Use
```

*Un objeto/colección compartido solo puede ser modificado por un proceso a la vez. **Use** bloquea el objeto/colección compartido de otros hilos, mientras que el último **End use** desbloquea todos los objetos y colecciones. Tratar de modificar un objeto/colección compartido sin al menos **Use...End use** genera un error. Cuando un proceso llama a **Use...End use** en un objeto/colección compartido que ya está siendo utilizado por otro proceso, simplemente se pone en espera hasta que el **End use** lo desbloquea (no se genera ningún error). Por lo tanto, las instrucciones en estructuras **Use...End use** deben ejecutarse rápidamente y desbloquear los elementos tan pronto como sea posible. Por lo tanto, se recomienda encarecidamente evitar la modificación de un objeto o colección compartido directamente desde la interfaz, por ejemplo, a través de un cuadro de diálogo. La asignación de objetos/colecciones compartidas a propiedades o elementos de otros objetos compartidos / colecciones está permitida y crea **grupos compartidos**. Un grupo compartido se crea automáticamente cuando un objeto compartido/colección se establece en un valor de propiedad o un objeto compartido. Los grupos compartidos permiten anidar objetos y colecciones compartidos pero se deben seguir reglas adicionales:*

- *La llamada de **Use** con un objeto/colección compartido de un grupo provocará el bloqueo de las propiedades/elementos de todos los objetos/colecciones del grupo.*
- *Un objeto o una colección compartida(o) solo puede pertenecer a un grupo compartido. Se devuelve un error si desea asignar un objeto/colección compartido ya agrupados a un grupo diferente.*
- *Los objetos/colecciones agrupados no se pueden desagrupar. Una vez incluido en un grupo compartido, un objeto o una colección compartida se une definitivamente al grupo durante todas la duración de la sesión. Incluso si todas las referencias del objeto/la colección se eliminen de los objetos/colecciones padre, permanecerán vinculados.*

Consulte el ejemplo 2 para ver una ilustración de las reglas de grupo compartidas.

***Nota:** los grupos compartidos se administran a través de una propiedad interna llamada locking identifier. Para obtener información detallada sobre este valor, consulte la sección avanzada **Identificador de bloqueo** a continuación.*

Read

*La lectura de propiedades o elementos de un objeto/colección compartido está permitida sin tener que llamar a la estructura **Use...End use**, incluso si el objeto/colección compartido está siendo utilizado por otro proceso.*

*Sin embargo, es necesario leer un objeto/colección compartido dentro de **Use...End use** cuando varios valores se vinculan entre sí y se deben leer a la vez, por razones de coherencia.*

Duplication

*Llamar a **OB Copy** con un objeto compartido (o con un objeto que contiene objetos compartidos como propiedades) es posible, pero devolverá un objeto estándar (no compartido) que incluye sus objetos contenidos (si corresponde) .*

Storage

Storage es un objeto compartido único, disponible automáticamente en cada aplicación y máquina. Este objeto compartido es devuelto por el comando ***Storage****. Está diseñado para referenciar los objetos/colecciones compartidos definidos durante la sesión que quiere que sean accesibles a todos los procesos, apropiativos o estándar.*

Tenga en cuenta que, a diferencia de los objetos compartidos estándar, el objeto Storage no crea un grupo compartido cuando los objetos/colección son agregados como sus propiedades. Esta excepción permite que el objeto Storage se use sin bloquear todos los objetos compartidos conectados.

Para más información, consulte la descripción del comando ***Storage****.*

Ejemplo 1

Desea iniciar varios procesos que realizan una tarea de inventario en diferentes productos y actualizar el mismo objeto compartido. El proceso principal ejemplifica un objeto compartido vacío y luego, inicia los otros procesos, pasando el objeto compartido y los productos para contar como parámetros:

```
ARRAY TEXT($_items;0)
... //fill the array with items to count
$nbItems:=Size of array($_items)
C_OBJECT($inventory)
$inventory:=New shared object
Use($inventory)
   $inventory.nbItems:=$nbItems
End use

 //Crear procesos
For($i;1;$nbItems)
   $ps:=New process("HowMany";0;"HowMany_"+$_items{$i};$_items{$i};$inventory)
 //$inventory object sent by reference
End for
```

En el método "HowMany", se hizo el inventario y el objeto compartido $inventory se actualiza tan pronto como es posible:

```
C_TEXT($1)
C_TEXT($what)
C_OBJECT($2)
C_OBJECT($inventory)
$what:=$1  //para mayor lecturabilidad
$inventory:=$2

$count:=CountMethod($what) //método para contar productos
Use($inventory) //utilizar objeto compartido
   $inventory[$what]:=$count  //guardar los resultados para este elemento
End use
```

Ejemplo 2

Los siguientes ejemplos resaltan reglas específicas al manejar ***grupos compartidos****:*

```
$ob1:=New shared object
$ob2:=New shared object
Use($ob1)
   $ob1.a:=$ob2  //se crea un primer grupo
End use

$ob3:=New shared object
$ob4:=New shared object
Use($ob3)
   $ob3.a:=$ob4  //se crea un segundo grupo
End use

Use($ob1) //utilización de un objeto del grupo 1
   $ob1.b:=$ob4  //ERROR
 //$ob4 ya pertenece a otro grupo
 //asignación no permitida
End use

Use($ob3)
   $ob3.a:=Null //elimina toda referencia a $ob4 del grupo 2
End use
```

```
Use($ob1) //utiliza un objeto de grupo 1
    $ob1.b:=$ob4   //ERROR
  //$ob4 aún pertenece a grupo 2
  //asignación no permitida
End use
```

Identificador de bloqueo

Nota: *esta sección ofrece información adicional sobre mecanismos internos específicos y puede ser útil en casos específicos únicamente.*

Todos los objetos/colecciones conectados:

- *El valor del identificador de bloqueo de un objeto/colección compartido "único" es* ***negativo****;*
- *El valor del identificador de bloqueo de un objeto/colección compartido "múltiple" es* ***positivo****.*

Cada nuevo objeto/colección compartido se crea con un identificador de bloqueo único (un valor entero largo). El valor inicial del identificador de bloqueo es siempre ***negativo****, lo que significa que el objeto/colección compartido es "single". Cuando un objeto/colección compartido con un estado "único" se establece como una propiedad o elemento de otro objeto/colección compartido, todos los objetos/colecciones conectados se convierten en "múltiples". Comparten el mismo identificador de bloqueo (propiedades/elementos agregados heredan el identificador de bloqueo del objeto/colección padre), y el valor del identificador de bloqueo se vuelve* ***positivo****.*
Cuando se elimina un objeto/colección compartido de su objeto/colección padre:

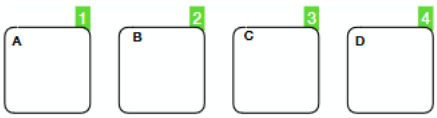
- *aún se consideran como "múltiples"*
- *mantienen el mismo identificador de bloqueo.*

Las reglas se aplican al establecer objetos/colecciones compartidos como propiedades o elementos a otros objetos/colecciones compartidos:

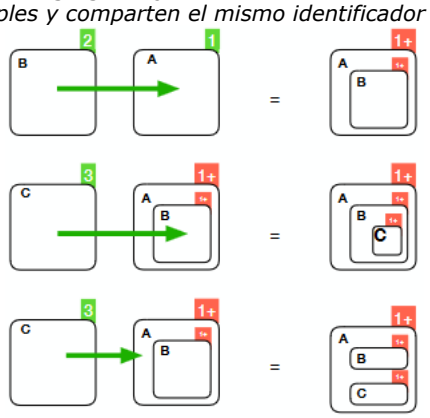
- *un objeto/colección "único" se puede unir a otro objeto/colección "único" o "múltiple",*
- *un objeto/colección "múltiple" se puede unir a otro objeto/ colección "múltiple" que tiene el mismo identificador de bloqueo,*
- *se produce un error si intenta adjuntar un objeto/colección "múltiple" a otro objeto/colección "único" o "múltiple" que no tiene el mismo identificador de bloqueo.*

Estas reglas se ilustran en los siguientes gráficos:

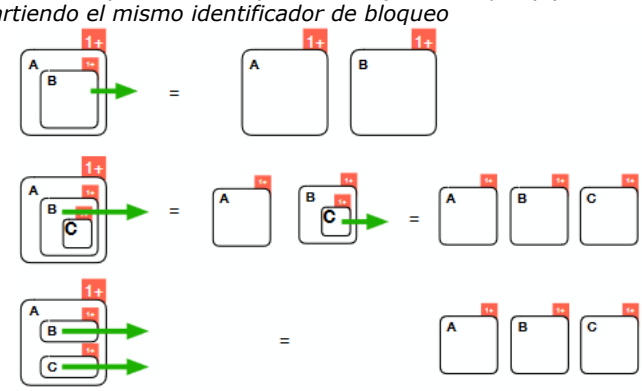
1. *Los objetos compartidos y las colecciones compartidas (A, B, C, D) se crean con un "identificador de bloqueo" interno y único (1, 2, 3, 4).*



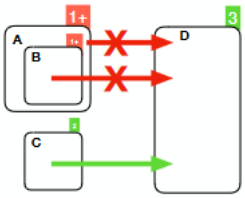
2. *Cuando se hace referencia a un único objeto (A) dentro de otro objeto único (B), ambos se vuelven múltiples y comparten el mismo identificador de bloqueo. Se pueden agregar objetos individuales adicionales a (A) ya sea al costado o dentro de (B). Todos los objetos se consideran múltiples y comparten el mismo identificador de bloqueo.*



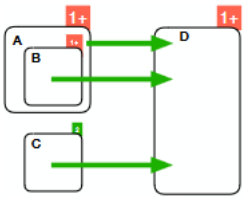
3. *Cuando un objeto múltiple (B) se elimina (desreferencia) de otro objeto múltiple (A), ambos permanecen múltiples y continúan conservando y compartiendo el mismo identificador de bloqueo*



4. *Los objetos múltiples (A, B) no se pueden usar como propiedades de un solo objeto (D), mientras que los objetos individuales (C) se pueden usar.*

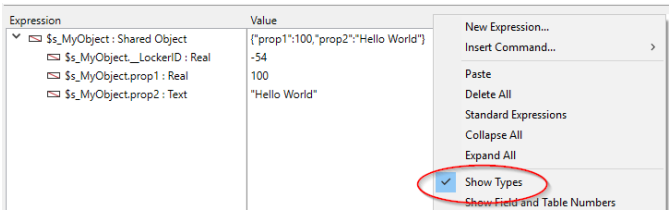


5. *Se pueden usar varios objetos (A, B) como propiedades del objeto (D) si comparten el mismo identificador de bloqueo. Los objetos individuales (C) también se pueden usar.*



Debugger

El identificador de bloqueo de objetos y colecciones compartidos se muestra en el depurador como la propiedad privada __LockerID. Puede mostrar el tipo de "Objeto compartido" en el depurador seleccionando ***Mostrar tipos*** *en el menú contextual del depurador:*



Tenga en cuenta que los objetos compartidos y las colecciones se pueden ingresar en el depurador, siempre que no se "usen" en ningún otro lugar, en cuyo caso no se pueden modificar.

New object

New object {(propiedad ; valor {; propiedad2 ; valor2 ; ... ; propiedadN ; valorN})} -> Resultado

Parámetro	Tipo		Descripción
propiedad	Texto	➡	Nombre de la propiedad a crear
valor	Texto, Fecha, Booleano, Puntero, Número, Objeto	➡	Valor de la propiedad
Resultado	Objeto	⮌	Nuevo objeto del lenguaje

Descripción

*El comando **New object** crea un nuevo objeto vacío o prellenado y devuelve su referencia.*

*Si no pasa ningún parámetro, **New object** crea un objeto vacío y devuelve su referencia. Debe asignar esta referencia a una variable 4D declarada con **C_OBJECT** o un campo objeto 4D.*

***Nota:** **C_OBJECT** declara una variable del tipo Objeto pero no crea ningún objeto.*

Opcionalmente, puede prefijar el nuevo objeto pasando uno o varios pares propiedad/valor como parámetros:

- *En el parámetro propiedad, pase la etiqueta de la propiedad a crear. Note que el parámetro propiedad es sensible a mayúsculas y minúsculas.*
- *En el parámetro valor, pase el valor que desea definir para la propiedad. Varios tipos de datos son soportados. Tenga en cuenta que:*
  - *Si pasa un puntero, se mantiene como está; Se evalúa utilizando el comando **JSON Stringify**,*
  - *Las fechas se almacenan en el forma de fecha "aaaa-mm-dd" o en cadena en formato "AAAA-MM-DDTHH:mm:ss.SSSZ" en función del parámetro actual relativo al almacenamiento de las fechas en los objetos (ver **Página Compatibilidad**). Al convertir las fechas 4D en texto antes de almacenarlas en el objeto, por defecto el programa toma en cuenta la zona horaria local. Puede modificar este comportamiento utilizando el selector Dates inside objects del comando **SET DATABASE PARAMETER**.*
  - *Si pasa una hora, se almacena como un número de milisegundos (Real).*

Ejemplo 1

Este comando puede crear objeto vacíos o llenos:

```
C_OBJECT($obj1)
C_OBJECT($obj2)
C_OBJECT($obj3)
$obj1:=New object
 // $obj1 = {}
$obj2:=New object("name";"Smith")
 // $obj2 = {name:Smith}
$obj3:=New object("name";"Smith";"age";40)
 // $obj3 = {name:Smith,age:40}
```

Ejemplo 2

Crear un nuevo objeto con un objeto como valor de parámetro:

```
C_OBJECT($Children;$Contact)

 //Crear un array objeto
ARRAY TEXT($arrChildren;3)
$arrChildren{1}:="Richard"
$arrChildren{2}:="Susan"
$arrChildren{3}:="James"
OB SET ARRAY($Children;"Children";$arrChildren)

 //Inicializar el objeto
$Contact:=New object("FirstName";"Alan";"LastName";"Parker";"age";30;"Children";$Children)
 // $Contact = {FirstName:Alan,LastName:Parker,age:30,Children:{Children:[Richard,Susan,James]}}
```

Ejemplo 3

Este comando es útil para pasar objetos como parámetros:

```
C_OBJECT($measures)
$measures:=Get database measures(New object("path";"DB.cacheReadBytes";"withHistory";True;"historyLength";120))
```

Ejemplo 4

Con este comando, puede fácilmente manejar objetos en bucles:

```
ARRAY OBJECT($refs;0)
C_LONGINT(vCounter)

For(vCounter;1;100)
    APPEND TO ARRAY($refs;New object("line";"Line number "+String(vCounter)))
End for
```

New shared object

New shared object {(propiedad ; valor {; propiedad2 ; valor2 ; ... ; propiedadN ; valorN})} -> Resultado

Parámetro	Tipo		Descripción
propiedad	Texto	➡	Nombre de la propiedad a crear
valor	Texto, Fecha, Booleano, Puntero, Número, Objeto	➡	Valor de la propiedad
Resultado	Objeto	↩	Nuevo objeto compartido

Descripción

*El comando **New shared object** crea un nuevo objeto compartido vacío o prellenado y devuelve su referencia. Para agregar o editar una propiedad a este objeto debe estar rodeado por la estructura **Use...End use**, de lo contrario, se devuelve un error. Sin embargo, es posible leer una propiedad fuera de una estructura **Use...End use**.*

***Nota:** para más información sobre objetos compartidos, consulte la página **Objetos y colecciones compartidos**.*

*Si no pasa ningún parámetro, **New shared object** crea un objeto vacío y devuelve su referencia. Debe asignar esta referencia a una variable 4D declarada con el comando **C_OBJECT**.*

***Nota:** **C_OBJECT** declara una variable del tipo Objeto pero no crea un objeto.*

Opcionalmente, puede rellenar el nuevo objeto pasando uno o varios pares de propiedad/valor como parámetros:

- *En el parámetro propiedad, pase la etiqueta de la propiedad que se creará (hasta 255 caracteres). Tenga en cuenta que el parámetro de propiedad es sensible a mayúsculas y minúsculas.*
- *En el parámetro valor, pase el valor que desea definir para la propiedad. Los objetos compartidos solo pueden contener valores de los siguientes tipos:*
  - *número (real, entero largo...) Los valores numéricos siempre se almacenan como reales.*
  - *texto*
  - *booleano*
  - *fecha*
  - *hora (almacenado como número de milisegundos - real)*
  - *null*
  - *objeto compartido(*)*
  - *colección compartida(*)*

  ***Nota:** a diferencia de los objetos estándar (no compartidos), los objetos compartidos no son compatibles con imágenes, punteros y objetos o colecciones que no se comparten.*

  *(*) Cuando se agrega un objeto o una colección compartido a un objeto compartido, comparten el mismo identificador de bloqueo. Para más información sobre este punto, consulte la sección **Identificador de bloqueo**.*

Ejemplo 1

Usted desea crear un nuevo objeto compartido prellenado:

```
C_OBJECT($contact)
$contact:=New shared object("name";"Smith";"firstname";"John")
```

Ejemplo 2

Usted desea crear y modificar un objeto compartido. La estructura debe llamarse para este objeto:

```
C_OBJECT($s_obj)
$s_obj:=New shared object("prop1";"alpha")
Use($s_obj)
   $s_obj.prop1:="omega"
End use
```

OB Copy

OB Copy (objeto {; resuelvePunt}) -> Resultado

Parámetro	Tipo		Descripción
objeto	Objeto, Campo Objeto	➡	Objeto estructurado
resuelvePunt	Booleano	➡	True = resuelve los puntero, False o se omite = no resuelve punteros
Resultado	Objeto	⮌	Copia de objeto

Descripción

*El comando **OB Copy** devuelve un objeto que contiene una copia completa de las propiedades, sub objetos y valores de objeto.*

*objeto debe haber sido definido utilizando el comando **C_OBJECT** o designar un campo objeto 4D.*

*Si el objeto contiene valores de tipo de puntero, por defecto la copia también contiene los punteros. Sin embargo, puede resolver los punteros al momento de la copia pasando **True** en el parámetro resuelvePunt. IEn este caso, cada puntero presente como valor en objeto se evalúa al momento de la copia y se utiliza su valor desreferenciado.*

***Nota:** si las propiedades del objeto son objetos compartidos o colecciones compartidas, se convierten en objetos o colecciones estándar (no compartidos) en la copia devuelta.*

Ejemplo 1

Usted quiere duplicar un objeto que contiene valores simples:

```
C_OBJECT($Object)
C_TEXT($JsonString)

ARRAY OBJECT($arraySel;0)
ALL RECORDS([Product])
While(Not(End selection([Product])))
   OB SET($Object;"id";[Product]ID_Product)
   OB SET($Object;"Product Name";[Product]Product_Name)
   OB SET($Object;"Price";[Product]Price)
   OB SET($Object;"Tax rate";[Product]Tax_rate)
   $ref_value:=OB Copy($Object) //direct copy
   APPEND TO ARRAY($arraySel;$ref_value)
 //el contenido de $ref_value es idéntico al de $Object
   NEXT RECORD([Product])
End while
 //el contenido de $ref_value
$JsonString:=JSON Stringify array($arraySel)
```

Ejemplo 2

Duplique un objeto que contenga punteros:

```
C_OBJECT($ref)

OB SET($ref;"name";->[Company]name) //object with pointers
OB SET($ref;"country";->[Company]country)
ARRAY OBJECT($sel;0)
ARRAY OBJECT($sel2;0)

ALL RECORDS([Company])

While(Not(End selection([Company])))
   $ref_comp:=OB Copy($ref) // copy without evaluating pointers
 // $ref_comp={"name":"->[Company]name","country":"->[Company]Country"}
   $ref_comp2:=OB Copy($ref;True) //copy with evaluation of pointers
 // $ref_comp={"name":"4D SAS","country":"France"}
   APPEND TO ARRAY($sel;$ref_comp)
   APPEND TO ARRAY($sel2;$ref_comp2)
   NEXT RECORD([Company])
End while

$Object:=JSON Stringify array($sel)
```

```
$Object2:=JSON Stringify array($sel2)

 // $Object = [{"name":"","country":""},{"name":"","country":""},...]
 // $Object2 = [{"name":"4D SAS","country":"France"},{"name":"4D, Inc","country":"USA"},{"name":"Catalan","country":"France"}...]
```

OB Get

OB Get (objeto ; propiedad {; tipo}) -> Resultado

Parámetro	Tipo		Descripción
objeto	Objeto, Campo Objeto	➡	Objeto estructurado
propiedad	Texto	➡	Nombre de la propiedad a leer
tipo	Entero largo	➡	Tipo al cual convertir el valor
Resultado	Expresión	⬅	Valor actual de propiedad

Descripción

*El comando **OB Get** devuelve el valor actual de la propiedad del objeto, convertido opcionalmente en el tipo definido.*

*objeto debe haber sido definido con el comando **C_OBJECT** o designar un campo objeto 4D.*

***Nota:** este comando soporta definiciones de atributo en objetos 4D Write Pro, como el comando **WP GET ATTRIBUTES** (ver el ejemplo 9).*

En el parámetro propiedad, pase la etiqueta de la propiedad a leer. Tenga en cuenta que el parámetro propiedad es sensible a mayúsculas y minúsculas

*Por defecto, 4D devuelve el valor de la propiedad en su tipo original. Puede "forzar" la escritura del valor devuelto utilizando el parámetro opcional tipo. Para ello, en tipo pase una de las siguientes constantes que se encuentran en el tema **Tipos de campos y variables**:*

Constante	Tipo	Valor
Is Boolean	Entero largo	6
Is collection	Entero largo	42
Is date	Entero largo	4
Is longint	Entero largo	9
Is null	Entero largo	255
Is object	Entero largo	38
Is picture	Entero largo	3
Is pointer	Entero largo	23
Is real	Entero largo	1
Is text	Entero largo	2
Is time	Entero largo	11

El comando devuelve el valor de la propiedad. Varios tipos de datos están soportados. Tenga en cuenta que:

- *un puntero se devuelve tal cual, puede ser evaluado utilizando el comando **JSON Stringify**,*
- *dependiendo de la configuración de la fecha de la base, las fechas en los atributos objeto se almacenan con el tipo de fecha o el tipo de texto (a partir de 4D v16 R6). Para más información, consulte la opción "Utilizar tipo de fecha en lugar del formato de fecha ISO en objetos" en **Página Compatibilidad**. Para que **OB Get** interprete correctamente una fecha almacenada como un texto, debe usar la constante Is date (ver ejemplo 5).*
- *en valores reales, el separador decimal es siempre un punto "."*
- *las horas se devuelven como un número. Las horas se almacenan en segundos por defecto en los objetos (ver nota de compatibilidad abajo). Note que **OB SET** almacena las horas en forma de milisegundos, conforme al estándar JavaScript, mientras 4D espera un número de segundos. Para una interpretación correcta por **OB Get** de una hora almacenada, utilice la constante Is null.*

Notas de compatibilidad:

- *(4D Write Pro) En versiones anteriores a la v17, las horas eran almacenadas en milisegundos en los objetos. Por razones de compatibilidad, este comportamiento anterior puede restablecerse con ayuda del selector Times inside objects del comando **SET DATABASE PARAMETER**. Cualquiera que sea el parámetro, el resultado será correcto cuando se utilice la constante Is time.*
- *(4D Write Pro) En versiones anteriores a v16 R6, cuando propiedad definía un atributo de imagen 4D Write Pro (como wk image), siempre se devolvía un valor de texto que contenía un URI de datos. A partir de 4D v16 R6, los atributos imagen 4D Write Pro siempre se devuelven como valores imagen. Debe usar una propiedad específica como wk image url ara obtener un URI de datos.*
- *En versiones anteriores a v16 R4, cuando propiedad contiene un valor nulo y si no se utiliza el parámetro tipo, 4D devuelve una cadena vacía. En 4D v16 R4 y superiores, la constante Is null se devuelve en este caso. Para conservar la compatibilidad, este cambio solo surte efecto si la opción "Utilizar la notación objetos para acceder a las propiedades de objetos (se requiere Unicode) " está habilitada en la base (ver el párrafo **Página Compatibilidad**).*

Ejemplo 1

Recuperación de un valor de tipo texto:

```
C_OBJECT($ref)
C_TEXT($FirstName)
OB SET($ref;"FirstName";"Harry")
$FirstName:=OB Get($ref;"FirstName") // $FirstName = "Harry" (text)
```

Ejemplo 2

Recuperación de un valor real convertido en entero largo:

```
OB SET($ref ;"age";42)
$age:=OB Get($ref ;"age") // $age es un número real (default)
$age:=OB Get($ref ;"age";ls longint) // $age es un entero largo
```

Ejemplo 3

Recuperación de los valores de un objeto:

```
C_OBJECT($ref1;$ref2)
OB SET($ref1;"LastName";"Smith") // $ref1={"LastName":"Smith"}
OB SET($ref2;"son";$ref1) // $ref2={"son":{"LastName":"Smith"}}
$son:=OB Get($ref2;"son") // $son={"LastName":"john"} (object)
$sonsName:=OB Get($son ;"name") // $sonsName="john" (text)
```

Ejemplo 4

Modificando de la edad de un empleado dos veces:

```
C_OBJECT($ref_john;$ref_jim)
OB SET($ref_john;"name";"John";"age";35)
OB SET($ref_jim;"name";"Jim";"age";40)
APPEND TO ARRAY($myArray;$ref_john) // creamos un objeto array
APPEND TO ARRAY($myArray;$ref_jim)
 // cambiamos la edad de John de 35 a 25
OB SET($myArray{1};"age";25)
 // cambiamos la edad de "John" en el array
For($i;1;Size of array($myArray))
    If(OB Get($myArray{$i};"name")="John")
        OB SET($myArray{$i};"age";36) // en cambio de 25
 // $ref_john={"name":"John","age":36}
    End if
End for
```

Ejemplo 5

Al recuperar una fecha, el valor resultante depende de la configuración actual de la fecha de la base.

- *Si la opción "Utilizar tipo fecha en lugar del formato de fecha ISO en objetos" no está seleccionada:*

```
C_OBJECT($object)
C_DATE($birthday)
C_TEXT($birthdayString)
OB SET($object;"Birthday";!30/01/2010!)
$birthday:=OB Get($object;"Birthday";ls date) //30/01/10
$birthdayString:=OB Get($object;"Birthday") //"2010-01-29T23:00:00.000Z" (Paris time zone)
```

- *Si la opción "Utilizar tipo fecha en lugar del formato de fecha ISO en objetos" está seleccionada:*

```
C_OBJECT($object)
C_DATE($birthday)
OB SET($object;"Birthday";!30/01/2010!)
$birthday:=OB Get($object;"Birthday") //30/01/10, no hay necesidad de ls date
```

Nota: *para más información sobre esta configuración, consulte* ***Página Compatibilidad****.*

Ejemplo 6

Utilización de objetos anidados:

```
C_OBJECT($ref1;$child;$children)
C_TEXT($childName)
```

```
OB SET($ref1;"firstname";"John";"lastname";"Monroe")
 //{"firstname":"john","lastname";"Monroe"}
OB SET($children;"children";$ref1)
$child:=OB Get($children;"children")
 //$son = {"firstname":"John","lastname":"Monroe"} (object)
$childName:=OB Get($child;"lastname")
 //$childName = "Monroe" (text)
 //o
$childName:=OB Get(OB Get($children;"children");"lastname")
 // $childName = "Monroe" (text)
```

Ejemplo 7

Recuperación en 4D de una hora almacenada en un objeto:

```
C_OBJECT($obj_o)
C_TIME($set_h;$get_h)

$set_h:=?01:00:00?+1
OB SET($obj_o;"myHour";$set_h)
 // $obj_o = {"myHour":3601}
 // La hora se almacena en segundos
$get_h:=OB Get($obj_o;"myHour";Is time)
 // $get_h = ?01:00:01?
```

Ejemplo 8

Ejemplos de manipulación de campos objeto 4D:

```
 // Definir un valor
OB SET([People]Identity_OB;"First name";$firstName)
OB SET([People]Identity_OB;"Last name";$lastName)

 // Obtener un valor
$firstName:=OB Get([People]Identity_OB;"First name")
$lastName:=OB Get([People]Identity_OB;"Last name")
```

Ejemplo 9

En un método formulario, puede escribir:

```
If(Form event=On Validate)
   OB SET([MyDocuments]My4DWP;"myatt_Last edition by";Current user)
   OB SET([MyDocuments]My4DWP;"myatt_Category";"Memo")
End if
```

También puede leer los atributos personalizados de los documentos:

```
vAttrib:=OB Get([MyDocuments]My4DWP;"myatt_Last edition by")
```

Ejemplo 10

Desea conocer el tamaño de una imagen almacenada en un atributo objeto:

```
C_LONGINT($vSize)
$vSize:=Picture size(OB Get($object;"photo";Is picture))
```

Nota: *si asigna el resultado del comando a una variable imagen, la constante Is picture no es necesaria. Ejemplo:*

```
C_PICTURE($vPict)
$vPict:=OB Get($object;"photo") //"is picture" es inútil en este caso
```

OB GET ARRAY

OB GET ARRAY (objeto ; propiedad ; array)

Parámetro	Tipo		Descripción
objeto	Objeto, Campo Objeto	➡	Objeto estructurado
propiedad	Texto	➡	Nombre de la propiedad a leer
array	Array texto, Array real, Array booleano, Array objeto, Array puntero, Array entero largo	⬅	Array valor de la propiedad

Descripción

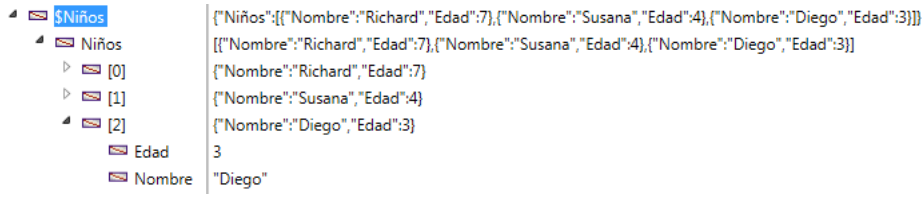
*El comando **OB GET ARRAY** recupera en array, el array de valores almacenados en la propiedad del objeto de lenguaje designado por el parámetro objeto.*

*objeto debe haber sido definido con el comando **C_OBJECT** o designar un campo objeto 4D.*

En el parámetro propiedad, pase la etiqueta de la propiedad a leer. Tenga en cuenta que el parámetro propiedad tiene en cuenta las mayúsculas y minúsculas.

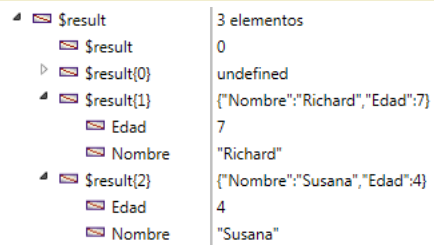
Ejemplo 1

*Dado un objeto array definido en el ejemplo del comando **OB SET ARRAY**:*



Queremos recuperar estos valores:

```
ARRAY OBJECT($result;0)
OB GET ARRAY($Children;"Children";$result)
```



Ejemplo 2

Queremos cambiar un valor en el primer elemento del array:

```
// Cambiar el valor de "age":
ARRAY OBJECT($refs)
OB GET ARRAY($refEmployees;"__ENTITIES";$refs)
OB SET($refs{1};"age";25)
```

OB GET PROPERTY NAMES

OB GET PROPERTY NAMES (objeto ; nomProp {; arrTipos})

Parámetro	Tipo		Descripción
objeto	Objeto	➡	Objeto estructurado
nomProp	Array texto	⬅	Nombres de las propiedades
arrTipos	Array entero largo	⬅	Tipos de propiedades

Descripción

*El comando **OB GET PROPERTY NAMES** devuelve, en arrProp, los nombres de las propiedades contenidas en el objeto de lenguaje designados por el parámetro objeto .*

*objeto debe haber sido definido utilizando el comando **C_OBJECT** o designar un campo objeto 4D.*

Pase un array texto en el parámetro arrProp. Si el array no existe, el comando lo crea y lo redimensiona de forma automática.

*Opcionalmente, también puede pasar un array entero largo en arrTipos. Para cada elemento de arrProp, el comando devuelve, en arrTipos, el tipo de valor almacenado en la propiedad. Puede comparar los valores recibidos con las siguientes constantes, que se encuentran en el tema "**Tipos de campos y variables**":*

Constante	Tipo	Valor
Is Boolean	Entero largo	6
Is collection	Entero largo	42
Is null	Entero largo	255
Is object	Entero largo	38
Is real	Entero largo	1
Is text	Entero largo	2
Object array	Entero largo	39

***Nota:** para los atributos array, el comando devuelve Is collection.*

Ejemplo 1

Desea probar que un objeto no está vacío:

```
ARRAY TEXT(arrNames;0)
ARRAY LONGINT(arrTypes;0)
C_OBJECT($ref_richard)
OB SET($ref_richard;"name";"Richard";"age";7)
OB GET PROPERTY NAMES($ref_richard;arrNames;arrTypes)
 // arrNames{1}="name", arrNames{2}="age"
 // arrTypes{1}=2, arrTypes{2}=1
If(Size of array(arrNames)#0)
 // ...
End if
```

Ejemplo 2

Utilizando un elemento de array de objetos:

```
C_OBJECT($Children;$ref_richard;$ref_susan;$ref_james)
ARRAY OBJECT($arrayChildren;0)

OB SET($ref_richard;"name";"Richard";"age";7)
APPEND TO ARRAY($arrayChildren;$ref_richard)
OB SET($ref_susan;"name";"Susan";"age";4;"girl";True) //atributo adicional
APPEND TO ARRAY($arrayChildren;$ref_susan)
OB SET($ref_james;"name";"James")
OB SET NULL($ref_james;"age") //null attribute
APPEND TO ARRAY($arrayChildren;$ref_james)

OB GET PROPERTY NAMES($arrayChildren{1};$arrNames;$arrTypes)
 // $arrayChildren{1} = {"name":"Richard","age":7}
 // $arrNames{1}="name"
 // $arrNames{2}="age"
 // $arrTypes{1}=2
 // $arrTypes{2}=1
```

```
OB GET PROPERTY NAMES($arrayChildren{2};$arrNames;$arrTypes)
 // $arrayChildren{3} = {"name":"Susan","age":4,"girl":true}
 // $arrNames{1}="name"
 // $arrNames{2}="age"
 // $arrNames{3}="girl"
 // $arrTypes{1}=2
 // $arrTypes{2}=1
 // $arrTypes{3}=6

OB GET PROPERTY NAMES($arrayChildren{3};$arrNames;$arrTypes)
 // $arrayChildren{3} = {"name":"James","age":null}
 // $arrNames{1}="name"
 // $arrNames{2}="age"
 // $arrTypes{1}=2
 // $arrTypes{2}=255
```

OB Get type

OB Get type (objeto ; propiedad) -> Resultado

Parámetro	Tipo		Descripción
objeto	Objeto	➡	Objeto estructurado
propiedad	Texto	➡	Nombre de la propiedad
Resultado	Entero largo	⮌	Tipo de valor de la propiedad

Descripción

*El comando **OB Get type** devuelve el tipo del valor asociado a la propiedad del objeto.*

*objeto debe haber sido definido utilizando el comando **C_OBJECT** o designar un campo objeto 4D.*

En el parámetro propiedad, pase la etiqueta de la propiedad cuyo tipo quiere conocer.
Tenga en cuenta que el parámetro propiedad tiene en cuenta las mayúsculas y minúsculas.

*El comando devuelve un entero largo que indica el tipo de valor. Puede comparar este valor con las siguientes constantes, que se encuentran en el tema "**Tipos de campos y variables**":*

Constante	Tipo	Valor
Is Boolean	Entero largo	6
Is collection	Entero largo	42
Is date	Entero largo	4
Is null	Entero largo	255
Is object	Entero largo	38
Is real	Entero largo	1
Is text	Entero largo	2
Is undefined	Entero largo	5

***Nota:** para los atributos imagen, el comando devuelve Is object.*

Ejemplo

Queremos obtener el tipo de valores estándar:

```
C_OBJECT($ref)
OB SET($ref;"nombre";"smith";"edad";42)
$type:=OB Get type($ref;"nombre") //$type devuelve 2
$type2:=OB Get type($ref;"edad") //$type2 devuelve 1
```

OB Is defined

OB Is defined (objeto {; propiedad}) -> Resultado

Parámetro	Tipo		Descripción
objeto	Objeto, Campo Objeto	➡	Objeto estructurado
propiedad	Texto	➡	Si se pasa = propiedad a verificar, si se omite = verificar el objeto
Resultado	Booleano	⮌	Si propiedad se omite: True si objeto está definido, de lo contrario False. Si propiedad se pasa: True si propiedad está definida, de lo contrario False

Descripción

El comando ***OB Is defined*** *devuelve* ***True*** *si objeto o propiedad se definen y* ***False*** *en caso contrario.*

objeto debe haber sido creado con el comando ***C_OBJECT*** *o designar un campo objeto 4D.*

Por defecto, si se omite el parámetro propiedad, el comando comprueba que objeto esté definido. Un objeto está definido si su contenido se ha sido inicializado.

Nota*: un objeto puede estar definido, pero vacío. Para saber si un objeto no está definido o está vacío, utilice el comando* ***OB Is empty****.*

Si pasa el parámetro propiedad, el comando comprueba si existe esta propiedad en objeto. Tenga en cuenta que el parámetro propiedad es sensible a las mayúsculas y minúsculas.

Ejemplo 1

Sintaxis probando la inicialización de un objeto:

```
C_OBJECT($object)
$def:=OB Is defined($object) // $def=false since $object no está incializado

OB SET($object;"Name";"Martin")
OB REMOVE($object;"Name")
$def2:=OB Is defined($object) // $def2=true ya que $object está vacío {} pero ha sido inicializado
```

Ejemplo 2

Prueba de la existencia de una propiedad:

```
C_OBJECT($ref)
OB SET($ref;"name";"smith";"age";42)
 //...
If(OB Is defined($ref;"age"))
 //...
End if
```

Esta prueba es equivalente a:

```
If(OB Get type($Object;"name")#Is undefined)
```

OB Is empty

OB Is empty (objeto) -> Resultado

Parámetro	Tipo		Descripción
objeto	Objeto, Campo Objeto	→	Objeto estructurado
Resultado	Booleano	↩	True si objeto está vacío o indefinido, de lo contrario False

Descripción

*El comando **OB Is empty** devuelve **True** si objeto no está definido o está vacío, y **False** si objeto está definido (inicializado) y contiene al menos una propiedad.*

*objeto debe haber sido creado con el comando **C_OBJECT** o designar un campo objeto 4D.*

Ejemplo

*Estos son los diferentes resultados de este comando como también del comando **OB Is defined**, dependiendo del contexto:*

```
C_OBJECT($ref)
$empty:=OB Is empty($ref) // True
$def:=OB Is defined($ref) // False

OB SET($ref;"nombre";"Susie";"edad";4)
 // $ref="{"nombre":"Susie","edad":4}"
$empty:=OB Is empty($ref) // False
$def:=OB Is defined($ref) // True

OB REMOVE($ref;"nombre")
OB REMOVE($ref;"edad")
$empty:=OB Is empty($ref) // True
$def:=OB Is defined($ref) // True
```

OB REMOVE

OB REMOVE (objeto ; propiedad)

Parámetro	Tipo		Descripción
objeto	Objeto, Campo Objeto	➡	Objeto estructurado
propiedad	Texto	➡	Nombre de la propiedad a eliminar

Descripción

*El comando **OB REMOVE** elimina la propiedad del objeto de lenguaje designado por el parámetro objeto. Este comando elimina la propiedad, así como su valor actual.*

*objeto debe haber sido definido utilizando el comando **C_OBJECT** o designar un campo objeto 4D.*

En el parámetro propiedad, pase la etiqueta de la propiedad a leer. Tenga en cuenta que el parámetro propiedad tiene en cuenta las mayúsculas y minúsculas.

Ejemplo

Usted quiere eliminar la propiedad "edad" de un objeto:

```
C_OBJECT($Object)
OB SET($Object;"nombre";"smith";"edad";42;"cliente";True)
 // $Object={"nombre":"smith","edad":42,"client":true}
OB REMOVE($Object;"edad")
 // $Object={"nombre":"smith","cliente":true}
```

OB SET

OB SET (objeto ; propiedad ; valor {; propiedad2 ; valor2 ; ... ; propiedadN ; valorN})

Parámetro	Tipo		Descripción
objeto	Campo Objeto, Objeto	➡	Objeto estructurado
propiedad	Texto	➡	Nombre de la propiedad a configurar
valor	Expresión	➡	Nuevo valor de la propiedad

Descripción

*El comando **OB SET** crea o modifica uno o más pares de propiedad/valor en el objeto de lenguaje designado por el parámetro objeto.*

*objeto debe haber sido definido usando el comando **C_OBJECT** o designar un campo objeto 4D.*
En el parámetro propiedad, pase la etiqueta de la propiedad a crear o modificar. Si la propiedad ya existe en objeto, su valor se actualiza. Si no existe, se crea.

Tenga en cuenta, que el parámetro propiedad es sensible a las mayúsculas y minúsculas.
En el parámetro valor, pase el valor que desea definir para la propiedad. Se admiten varios tipos de datos. Tenga en cuenta que:

- *si pasa un puntero, se mantiene como es, se evalúa utilizando el comando **JSON Stringify***
- *si pasa una fecha, se almacenará con el tipo de fecha o como texto en formato ISO dependiendo de la configuración actual de la fecha de la base. Para más información, consulte la opción "Utilizar tipo de fecha en lugar de formato de fecha ISO en objetos" en **Página Compatibilidad**.*
- *si pasa una hora, se almacena en forma de un número de segundos (real) en objeto*
- *si pasa un objeto de lenguaje o una colección, el comando utiliza una referencia y no una copia. Toda modificación aplicada al objeto o colección se informará a todas las referencias*
- *a partir de 4D v16 R4, puede pasar una imagen de todo tipo soportado (ver **Formatos nativos soportados**).*

Ejemplo 1

Creación de un objeto y adición de una propiedad de tipo texto:

```
C_OBJECT($Object)
OB SET($Object ;"FirstName";"John";"LastName";"Smith")
 // $Object = {"FirstName":"John","LastName":"Smith"}
```

Ejemplo 2

Creación de un objeto y adición de una propiedad de tipo booleano:

```
C_OBJECT($Object)
OB SET($Object ;"LastName";"smith";"age";42;"client";True)
 // $Object = {"LastName":"smith","age":42,"client":true}
```

Ejemplo 3

Modificación de una propiedad:

```
 // $Object = {"FirstName":"John","LastName":"Smith"}
OB SET($Object ;"FirstName";"Paul")
 // $Object = {"FirstName":"Paul","LastName":"Smith"}
```

Ejemplo 4

Adición de una propiedad:

```
 // $Object = {"FirstName":"John","LastName":"Smith"}
OB SET($Object ;"department";"Accounting")
 // $Object = {"FirstName":"Paul","LastName":"Smith","department":"Accounting"}
```

Ejemplo 5

Renombrando una propiedad:

```
C_OBJECT($Object)
OB SET($Object ;"LastName";"James";"age";35)
 // $Object = {"LastName":"James","age":35}
OB SET($Object ;"FirstName";OB Get($Object ;"LastName"))
 // $Object = {"FirstName":""James","nom":"James","age":35}
OB REMOVE($Object ;"LastName")
 // $Object = {"FirstName":""James","age":35}
```

Ejemplo 6

Usando un puntero:

```
 // $Object = {"FirstName":"Paul","LastName":"Smith"}
C_TEXT($LastName)
OB SET($Object ;"LastName";->$LastName)
 // $Object = {"FirstName":"Paul","LastName":"->$LastName"}
$JsonString:=JSON Stringify($Object)
 // $JsonString="{"FirstName":"Paul","LastName":""}
$LastName:="Wesson"
$JsonString:=JSON Stringify($Object)
 // $JsonString="{"FirstName":"Paul","LastName":"Wesson"}
```

Ejemplo 7

Usando un objeto:

```
C_OBJECT($ref_smith)
OB SET($ref_smith ;"name";"Smith")
C_OBJECT($ref_emp)
OB SET($ref_emp ;"employee";$ref_smith)
$Json_string :=JSON Stringify($ref_emp)
 // $ref_emp = {"employee":{"name":"Smith"}} (object)
 // $Json_string = "{"employee":{"name":"Smith"}}" (string)
```

También puede cambiar un valor sobre la marcha:

```
OB SET($ref_smith ;"name";"Smyth")
 // $ref_smith = {"employee":{"name":"Smyth"}}
$string:=JSON Stringify($ref_emp)
 // $string = "{"employee":{"name":"Smyth"}}"
```

Ejemplo 8

Si ha definido el campo [Rect]Desc como un campo objeto, puede escribir:

```
CREATE RECORD([Rect])
[Rect]Name:="Blue square"
OB SET([Rect]Desc;"x";"50";"y";"50";"color";"blue")
SAVE RECORD([Rect])
```

Ejemplo 9

*Usted quiere exportar datos en JSON que contienen una fecha 4D que desea convertir en una cadena sin información de zona horaria. Note que la conversión ocurre cuando la fecha se guarda en el objeto, debe llamar al comando **SET DATABASE PARAMETER** antes de llamar a **OB SET**:*

```
C_OBJECT($o)
$vDateSetting:=Get database parameter(Dates inside objects) //guardar la configuración actual
SET DATABASE PARAMETER(Dates inside objects;String type without time zone)
OB SET($o ;"myDate";Current date) // conversión JSON
$json:=JSON Stringify($o)
SET DATABASE PARAMETER(Dates inside objects;$vDateSetting)
```

Ejemplo 10

En un método formulario, puede escribir:

```
If(Form event=On Validate)
    OB SET([MyDocuments]My4DWP;"myatt_Last edition by";Current user)
    OB SET([MyDocuments]My4DWP;"myatt_Category";"Memo")
End if
```

También puede leer los atributos personalizados de los documentos:

```
vAttrib:=OB Get([MyDocuments]My4DWP;"myatt_Last edition by")
```

Ejemplo 11

Usted desea definir una colección como un valor propiedad. Puede escribir:

```
C_OBJECT($person)
C_COLLECTION($myCol)

$person:=OB New
$myCol:=New collection("Mike";25;"Denis";12;"Henry";4;True)
OB SET($person;"Name";"Jones";"Children";$myCol)
```

Ejemplo 12

Usted desea almacenar una imagen en un campo objeto. Puede escribir:

```
C_PICTURE($vPict)
READ PICTURE FILE("photo.jpg";$vPict)
If(OK=1)
    OB SET([Emp]Children;"photo";$vPict)
End if
```

OB SET ARRAY

OB SET ARRAY (objeto ; propiedad ; array)

Parámetro	Tipo		Descripción
objeto	Campo Objeto, Objeto	➡	Objeto estructurado
propiedad	Texto	➡	Nombre de la propiedad a definir
array	Array	➡	Array a almacenar en propiedad

Descripción

*El comando **OB SET ARRAY** define el array a asociarse a la propiedad en el objeto definido por el parámetro objeto.*
*objeto debe haber sido definido con el comando **C_OBJECT** o designar un campo objeto 4D.*

En el parámetro propiedad, pase la etiqueta de la propiedad a crear o modificar. Si la propiedad ya existe en objeto, su valor se actualiza. Si no existe, se crea.
Tenga en cuenta que el parámetro propiedad tiene en cuenta las mayúsculas y minúsculas.

En el parámetro array, pase el array que se debe pasar como valor de la propiedad. Se admiten varios tipos de array: real, entero largo, texto, booleano, objeto, puntero. Los arrays imagen se soportan a partir de 4D v16 R4.

***Nota**: no es posible utilizar arrays de dos dimensiones.*

Ejemplo 1

Utilizando un array texto:

```
C_OBJECT($Children)
ARRAY TEXT($arrChildren;3)
$arrChildren{1}:="Richard"
$arrChildren{2}:="Susan"
$arrChildren{3}:="James"

OB SET ARRAY($Children;"Children";$arrChildren)
 // Valor de $Children = {"Children":["Richard","Susan","James"]}
```

Ejemplo 2

Adición de un elemento de un array:

```
ARRAY TEXT($arrText;2)
$arrText{1}:="Smith"
$arrText{2}:="White"
C_OBJECT($Employees)
OB SET ARRAY($Employees;"Employees";$arrText)
APPEND TO ARRAY($arrText;"Brown") // Añadir el array 4D
 // $Employees = {"Employees":["Smith","White"]}

OB SET ARRAY($Employees;"Employees";$arrText)
 // $Employees = {"Employees":["Smith","White","Brown"]}
```

Ejemplo 3

Utilizando una array texto con selección de un elemento:

```
 // $Employees = {"Employees":["Smith","White","Brown"]}
OB SET ARRAY($Employees ;"Manager";$arrText{1})
 // $Employees = {"Employees":["Smith","White","Brown"],"Manager":["Smith"]}
```

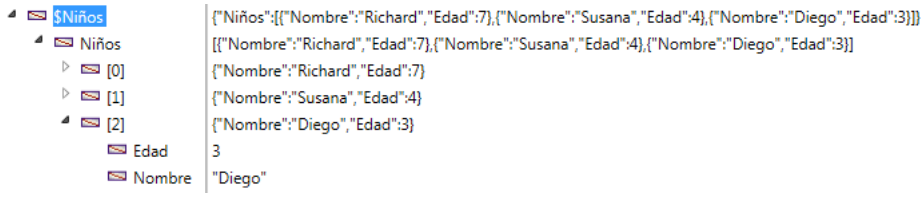
Ejemplo 4

Uso de un array objeto:

```
C_OBJECT($Children;$ref_richard;$ref_susan;$ref_james)
ARRAY OBJECT($arrChildren;0)
```

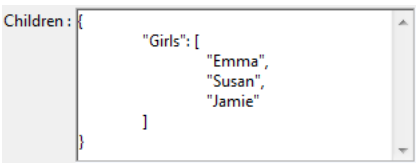
```
OB SET($ref_richard;"nom";"Richard";"age";7)
APPEND TO ARRAY($arrChildren;$ref_richard)
OB SET($ref_susan;"name";"Susan";"age";4)
APPEND TO ARRAY($arrChildren;$ref_susan)
OB SET($ref_james;"name";"James";"age";3)

APPEND TO ARRAY($arrChildren;$ref_james)

 // $arrChildren {1} = {"name":"Richard","age":7}
 // $arrChildren {2} = {"name":"Susan","age":4}
 // $arrChildren {3} = {"name":"James","age":3}

OB SET ARRAY($Children;"Children";$arrChildren)

 // $Children = {"Children":[{"name":"Richard","age":7},{"name":"Susan",
 // "age":4},{"name":"James","age":3}]}
```

Así es como el objeto aparece en el depurador:



Ejemplo 5

```
ARRAY TEXT($arrGirls;3)
$arrGirls{1}:="Emma"
$arrGirls{2}:="Susan"
$arrGirls{3}:="Jamie"
OB SET ARRAY([People]Children;"Girls";$arrGirls)
```



Ejemplo 6

Utilizando un array imagen:

```
ARRAY PICTURE($arrPhotos;3)
READ PICTURE FILE("pict1.jpg";$arrPhotos{1})
READ PICTURE FILE("pict2.jpg";$arrPhotos{2})
READ PICTURE FILE("pict3.jpg";$arrPhotos{3})

OB SET ARRAY([Cities]Places;"Photoset";$arrPhotos)
```

OB SET NULL

OB SET NULL (objeto ; property)

Parámetro	Tipo		Descripción
objeto	Objeto, Campo Objeto	➡	Objeto estructurado
property	Texto	➡	Nombre de la propiedad donde el valor nulo se va a aplicar

Descripción

*El comando **OB SET NULL** almacena el valor **null** en el objeto de lenguaje designado por el parámetro objeto .*

*objeto debe haber sido definido utilizando el comando **C_OBJECT** o designar un campo objeto 4D.*

*En el parámetro propiedad, pase la etiqueta de la propiedad en la que desea almacenar el valor **null**. Si la propiedad ya existe en objeto, su valor se actualiza. Si no existe, se crea.*
Tenga en cuenta que el parámetro propiedad tiene en cuenta las mayúsculas y minúsculas.

Ejemplo

Queremos poner el valor nulo en la propiedad "edad" para Lea:

```
C_OBJECT($ref)
OB SET($ref;"nombre";"Lea";"edad";4)
  // $ref = {"nombre":"Lea","edad":4}
...
OB SET NULL($ref ;"edad")
  // $ref = {"nombre":"Lea","edad":null}
```

Storage

Storage -> Resultado

Parámetro	Tipo		Descripción
Resultado	Objeto		Catálogo de objetos compartidos registrados y colecciones compartidas

Descripción

*El método **Storage** devuelve el catálogo de objetos compartidos o colecciones compartidas registrados registradas en el objeto Storage en la máquina o componente actual.*

*El catálogo devuelto por **Storage** es creado automáticamente por 4D y está disponible para todos los procesos de la base, incluidos los procesos apropiativos. Existe un catálogo **Storage** por máquina y componente: en una aplicación cliente/servidor, hay un objeto compartido **Storage** en el servidor y un objeto compartido **Storage** en cada aplicación 4D remota; si la base usa componentes, hay un objeto **Storage** por componente.*

*Utilice el catálogo **Storage** para hacer referencia a cualquier objeto compartido o colección compartida que desee utilizar desde cualquier proceso apropiativo o estándar. Para registrar un objeto compartido o una colección compartida en el catálogo, agregue su referencia al objeto compartido devuelto por **Storage**.*

*Como el catálogo devuelto por **Storage** es un objeto compartido, sigue las reglas descritas en la sección **Objetos y colecciones compartidos**, pero con algunas características específicas:*

- *Este objeto solo puede contener objetos compartidos y colecciones compartidas. Intentar agregar otros tipos de valores (objetos no compartidos o colecciones, nulos, valores escalares) generará un error.*
- *Para agregar una propiedad a este objeto debe estar rodeado por la estructura **Use...End use**, de lo contrario, se devuelve un error. Sin embargo, es posible leer un atributo fuera de una estructura **Use...End use**.*
- *Cuando está rodeado por la estructura **Use...End use**, los atributos de primer nivel de **Storage** están bloqueados para otros procesos.*
- *A diferencia de los objetos compartidos estándar, el objeto devuelto por **Storage** NO compartirá su identificador de bloqueo con objetos compartidos o colecciones agregadas como atributos (para más información, consulte la sección **Identificador de bloqueo**).*

Ejemplo 1

*Una práctica común podría ser inicializar el objeto **Storage** en el **Método base On Startup** :*

```
Use(Storage)
    Storage.counters:=New shared object("customers";0;"invoices";0)
End use
```

Ejemplo 2

*Este ejemplo muestra una forma estándar de establecer valores de **Almacenamiento**:*

```
Use(Storage)
    Storage.mydata:=New shared object
    Use(Storage.mydata)
        Storage.mydata.prop1:="Smith"
        Storage.mydata.prop2:=100
    End use
End use
```

Ejemplo 3

***Storage** permite implementar un singleton con una inicialización lenta, como se muestra en el siguiente ejemplo.*
***Nota:** para más información acerca de los patrones de singleton, puede consultar esta página de Wikipedia.*

```
C_LONGINT($0)
C_LONGINT($counterValue)
C_OBJECT(counter) //crea una referencia al contador para el proceso

If(counter=Null) //Si esta referencia es nula, obtenga if de Storage
    Use(Storage) // ¡El uso del almacenamiento solo se necesita una vez!
        If(Storage.counter=Null)
            Storage.counter:=New shared object("operation";0)
        End if
        counter:=Storage.counter //Obtener la referencia del objeto compartido contador
    End use
```

```
End if
Use(counter) //use directamente el contador de objetos compartidos (¡no necesita usar Storage!)
    counter.operation:=counter.operation+1
    $counterValue:=counter.operation
End use

$0:=$counterValue
```

Operadores

- *Operadores*
- *Operadores de bits*
- *Operadores de comparación*
- *Operadores de fechas*
- *Operadores lógicos*
- *Operadores numéricos*
- *Operadores de imágenes*
- *Operadores de cadenas*
- *Operadores de horas*

Operadores

Los operadores son símbolos utilizados para especificar operaciones realizadas entre expresiones. Ellos:

- *Efectúan cálculos sobre números, fechas, y horas.*
- *Efectúan operaciones sobre cadenas, operaciones booleanas sobre expresiones lógicas y operaciones especializadas sobre imágenes.*
- *Combinan expresiones simples para generar nuevas expresiones.*

Precedencia

El orden en el cual una expresión se evalua se llama precedencia. 4D tiene una precedencia estricta de izquierda a derecha, en el cual el orden algebraíco no se aplica. Por ejemplo:

```
3+4*5
```

devuelve 35, porque la expresión se evalua como 3 + 4, que da 7, el cual se multiplica por 5, dando como resultado final 35.
Para hacer caso omiso de la precedencia izquierda derecha, usted DEBE utilizar paréntesis. Por ejemplo:

```
3+(4*5)
```

*devuelve 23 porque la expresión (4 * 5) se evalua primero, por el paréntesis. El resultado es 20, al cual se le añade 3 para dar como resultado final 23.*

Los paréntesis pueden ser incluidos dentro de otros paréntesis. Asegúrese de que cada paréntesis izquierdo tenga un paréntesis derecho correspondiente para que haya una evaluación correcta de las expresiones. La falta o uso incorrecto de los paréntesis puede provocar resultados inesperados o expresiones inválidas. Además, si tienen la intención de compilar sus aplicaciones, debe tener utilizar correctamente los paréntesis, el compilador interpretará como un error de sintáxis el hecho de que falte un paréntesis.

El operador de asignación

DEBE distinguir el operador de asignación := de los otros operadores. En lugar de combinar expresiones en una nueva, el operador de asignación copia el valor de la expresión a la derecha del operador de asignación en la variable o el campo a la izquierda del operador. Por ejemplo, la siguiente línea coloca el valor 4 (el número de caracteres en la palabra Acme) en la variable llamada ***MiVar****.* ***MiVar*** *se toma entonces como un valor numérico.*

```
MiVar := Length ("Acme")
```

Importante: *NO confunda el operador de asignación := con el operador de comparación de igualdad =.*

Los otros operadores del lenguaje 4D se describen en las siguientes secciones:

Operadores de cadenas

Ver la sección ***Operadores de cadenas****.*

Operadores numéricos

Ver la sección ***Operadores numéricos****.*

Operadores de fechas

Ver la sección ***Operadores de fechas****.*

Operadores de horas

Ver la sección ***Operadores de horas****.*

Operadores de comparación

Ver la sección ***Operadores de comparación****.*

Operadores lógicos

Ver la sección ***Operadores lógicos****.*

Operadores de imágenes

*Ver la sección **Operadores de imágenes**.*

Operadores de bits

*Ver la sección **Operadores de bits**.*

Operadores de bits

Los operadores de bits se aplican a expresiones o valores de tipo Entero largo.

***Nota:** si pasa un valor de tipo Entero o Numérico a un operador de bits, 4D lo convierte en Entero largo antes de calcular la expresión que utiliza el operador de bits.*

Cuando utilice operadores de bits, debe considerar un valor de tipo Entero largo como un array de 32 bits. Los bits se numeran de 0 a 31, de derecha a izquierda.

Como cada bit puede valer 0 o 1, usted puede igualmente considerar un valor de tipo Entero largo como una expresión donde puede almacenar 32 valores de tipo Booleano. Cuando el bit vale 1, el valor es True y cuando vale 0, el valor es False.

*Una expresión que utiliza un operador de bits devuelve un valor de tipo Entero largo, excepto para el operador **Bit Test**, donde la expresión devuelve un valor Booleano. La siguiente tabla lista los operadores de bits y su sintaxis:*

Operación	**Operador**	**Sintaxis**	**Devuelve**	
AND	&	E.largo & E.largo	E.largo	
OR (inclusivo)	|	E.largo | E.largo	E.largo	
OR (exclusivo)	^|	E.largo ^| E.largo	E.largo	
Left Bit Shift	<<	E.largo << E.largo	E.largo	(ver nota 1)
Right Bit Shift	>>	E.largo >> E.largo	E.largo	(ver nota 1)
Bit Set	?+	E.largo ?+ E.largo	E.largo	(ver nota 2)
Bit Clear	?-	E.largo?- E.largo	E.largo	(ver nota 2)
Bit Test	??	E.largo?? E.largo	Booleano	(ver nota 2)

Notas
*(1) En las operaciones utilizando **Left Bit Shift** y **Right Bit Shift**,el segundo operando indica el número de posiciones de bits que el primer operando se moverá en el valor resultante. Por lo tanto, este segundo operando debe estar entre 0 y 31. Note sin embargo, que mover 0 devuelve un valor sin cambio y mover más de 31 bits devuelve 0x00000000 porque todos los bits se pierden. Si pasa otro valor como segundo operando, el resultado no es significativo.*
*(2) En las operaciones utilizando **Bit Set**, **Bit Clear** y **Bit Test**, el segundo operando indica el número del bit en el cual actual. Por lo tanto, este segundo operando debe estar entre 0 y 31. De lo contrario, la expresión devuelve el valor del primer operando sin cambiar el valor por **Bit Set** y **Bit Clear**, y devuelve False para **Bit Test**.*

La siguiente tabla lista los operadores de bits y sus efectos:

Operación	Descripción
AND	Cada bit devuelto es el resultado de la operación AND lógica aplicada a los dos operandos. Esta es la tabla de AND lógica: 1 & 1 --> 1 0 & 1 --> 0 1 & 0 --> 0 0 & 0 --> 0 En otras palabras, el bit resultante es 1 si los dos bits operandos valen 1; de lo contrario el bit resultante es 0.
OR (inclusivo)	Cada bit resultante es el resultado de la operación OR inclusivo lógico aplicado a los bits en los dos operandos. Esta es la tabla del OR inclusivo lógico: 1 | 1 --> 1 0 | 1 --> 1 1 | 0 --> 1 0 | 0 --> 0 En otras palabras, el bit resultante es 1 si al menos uno de los dos bits operandos es 1; de lo contrario el bit resultante es 0.
O (exclusivo)	Cada bit devuelto es el resultado de la operación XOR exclusivo lógico aplicado a los dos bits operandos. Esta es la tabla del XOR exclusivo lógico: 1 ^| 1 --> 0 0 ^| 1 --> 1 1 ^| 0 --> 1 0 ^| 0 --> 0 En otras palabras, el bit resultante es 1 si sólo uno de los dos bits operandos es 1; de lo contrario el bit resultante es 0.
Left Bit Shift	El valor devuelto es definido para el valor del primer operando, luego los bits resultantes se mueven a la izquierda el número de posiciones indicadas por el segundo operando. Los bits a la izquierda se pierden y los nuevos bits a la derecha toman el valor 0. Nota: teniendo en cuenta sólo los valores positivos, moviendo a la izquierda N bits es lo mismo que multiplicar por 2^N.
Right Bit Shift	El valor resultante se asigna al primer operando, luego los bits resultantes se mueven a la derecha el número de posición indicada por el segundo operando. Los bits a la derecha se pierden y los nuevos bits a la izquierda toman el valor 0. Nota: teniendo en cuenta sólo los valores positivos, mover a la derecha N bits es lo mismo que dividir por 2^N.
Bit Set	El valor resultante es definido por el primer valor de operando, luego el bit resultante, cuyo número se indica por el segundo operando, toma el valor 1. Los otros bits no cambian.
Bit Clear	El valor devuelto es definido para el valor del primer operando, luego el bit resultante, cuyo número es indicado por el segundo operando, toma el valor 0. Los otros bits no cambian.
Bit Test	Devuelve True si, en el primer operando, el bit cuyo número es indicado por el segundo operando es igual a 1. Devuelve Falso si, en el primer operando, el bit cuyo número se indica por el segundo operando es igual a 0.

Ejemplo

(1) La siguiente tabla da un ejemplo de cada operador de bits:

Operación	Ejemplo	Resultado
Bitwise AND	0x0000FFFF & 0xFF00FF00	0x0000FF00
Bitwise OR (inclusivo)	0x0000FFFF | 0xFF00FF00	0xFF00FFFF
Bitwise OR (exclusivo)	0x0000FFFF ^| 0xFF00FF00	0xFF0000FF
Left Bit Shift	0x0000FFFF << 8	0x00FFFF00
Right Bit Shift	0x0000FFFF >> 8	0x000000FF
Bit Set	0x00000000 ?+ 16	0x00010000
Bit Clear	0x00010000 ?- 16	0x00000000
Bit Test	0x00010000 ?? 16	True

(2) 4D ofrece muchas constantes predefinidas. Los nombres de algunas de estas constantes comienzan por "bit" o "mask." Por ejemplo, este es el caso de las constantes del tema **Propiedades de los recursos**:

Constante	Tipo	Valor
Changed resource bit	Entero largo	1
Preloaded resource bit	Entero largo	2
Changed resource mask	Entero largo	2
Protected resource bit	Entero largo	3
Locked resource bit	Entero largo	4
Preloaded resource mask	Entero largo	4
Purgeable resource bit	Entero largo	5
System heap resource bit	Entero largo	6
Protected resource mask	Entero largo	8
Locked resource mask	Entero largo	16
Purgeable resource mask	Entero largo	32
System heap resource mask	Entero largo	64

*Estas constantes permiten probar el valor devuelto por **Get resource properties** o crear el valor a pasar a **SET RESOURCE PROPERTIES**. Las constantes cuyo nombre termina en "bit" dan la posición del bit que quiere probar, borrar, o definir. Las constantes cuyo nombre termina en "mask" dan un valor de tipo Entero largo donde sólo el bit (que quiere probar, borrar o asignar) es igual a uno.*

Por ejemplo, para probar si un recurso (cuyas propiedades han sido almacenadas en la variable $vlResAttr) es purgable o no, puede escribir:

```
If($vlResAttr ?? Purgeable resource bit) ` ¿El recurso es purgable?
```

o:

```
If(($vlResAttr & Purgeable resource mask)#0)El recurso es purgable?
```

Al contrario, puede utilizar estas constantes para definir el mismo bit. Puede escribir:

```
$vlResAttr:=$vlResAttr ?+Purgeable resource bit
```

o:

```
$vlResAttr:=$vlResAttr /Purgeable resource bit
```

*(3) Este ejemplo almacena dos valores **Enteros** en un valor Entero largo. Puede escribir:*

```
$vlLong:=($viIntA<<16)/$viIntB  ` Almacenar dos Enteros en un Entero largo

$vlIntA:=$vlLong>>16 ` Extraer el entero almacenado en high-word

$viIntB:=$vlLong & 0xFFFF ` Extraer el entero almacenado en low-word
```

***Consejo:** sea cuidadoso al manipular valores de tipo Entero largo o Entero con expresiones que combinan operadores numéricos y bitwise. El bit superior (bit 31 para Entero largo, bit 15 para Entero) define el signo del valor, positivo si es cero, negativo si es 1. Los operadores numéricos utilizan este bit para detectar el signo de un valor, pero los operadores de bits no tienen en cuenta el significado de este bit.*

Operadores de comparación

Las tablas en esta sección muestran los operadores de comparación. Estos operadores pueden ser aplicados a expresiones de tipo cadena, numérico, fecha, hora, puntero e imagen con metadatos (no puede utilizarlos con expresiones de tipo array o BLOB).
*Una expresión que utiliza un operador de comparación devuelve un valor booleano, **TRUE** o **FALSE**.*
***Nota:** es posible comparar dos imágenes utilizando el comando **Equal pictures**.*

Comparaciones de cadenas

Operación	Sintaxis	Devuelve	Expresión	Valor
Igualdad	Cadena= Cadena	Booleano	"abc" = "abc"	True
			"abc" = "abd"	False
Desigualdad	Cadena# Cadena	Booleano	"abc" # "abd"	True
			"abc" # "abc"	False
Mayor que	Cadena> Cadena	Booleano	"abd" > "abc"	True
			"abc" > "abc"	False
Menor que	Cadena< Cadena	Booleano	"abc" < "abd"	True
			"abc" < "abc"	False
Mayor que o igual a	Cadena>= Cadena	Booleano	"abd" >= "abc"	True
			"abc" >= "abd"	False
Menor que o igual a	Cadena<= Cadena	Booleano	"abc" <= "abd"	True
			"abd" <= "abc"	False
Contiene palabra	Cadena% Cadena	Booleano	"Alpha Bravo" % "Bravo"	True
			"Alpha Bravo" % "ravo"	False
	Imagen % Cadena	Booleano	Picture_expr % "Mer"	True (*)

***(*)** Si la palabra clave "Mer" está asociada a la imagen almacenada en la expresión imagen (campo o variable).*

***Importante:** al final de esta sección se ofrece información adicional sobre comparaciones de cadenas.*

Operadores numéricos

Operación	Sintaxis	Devuelve	Expresión	Valor
Igualdad	Número = Número	Booleano	10 = 10	True
			10 = 11	False
Desigualdad	Número # Número	Booleano	10 #11	True
			10 # 10	False
Mayor que	Número > Número	Booleano	11 > 10	True
			10 > 11	False
Menor que	Número < Número	Booleano	10 < 11	True
			11 < 10	False
Mayor que o igual a	Número >= Número	Booleano	11 >= 10	True
			10 >= 11	False
Menor que o igual a	Número <= Número	Booleano	10 <= 11	True
			11 <= 10	False

***Nota**: para más información sobre la precisión de las comparaciones de igualdad de números reales, consulte el comando **SET REAL COMPARISON LEVEL**.*

Comparación de fechas

Operación	Sintaxis	Devuelve	Expresión	Valor
Igualdad	Fecha= Fecha	Booleano	!1/1/97! =!1/1/97!	True
			!1/20/97! =!1/1/97!	False
Desigualdad	Fecha# Fecha	Booleano	!1/20/97! # !1/1/97!	True
			!1/1/97! # !1/1/97!	False
Mayor que	Fecha> Fecha	Booleano	!1/20/97! > !1/1/97!	True
			!1/1/97! > !1/1/97!	False
Menor que	Fecha< Fecha	Booleano	!1/1/97! < !1/20/97!	True
			!1/1/97! < !1/1/97!	False
Mayor que o igual a	Fecha>= Fecha	Booleano	!1/20/97! >=!1/1/97!	True
			!1/1/97!>=!1/20/97!	False
Menor que o igual a	Fecha<= Fecha	Booleano	!1/1/97!<=!1/20/97!	True
			!1/20/97!<=!1/1/97!	False

Comparaciones de horas

Operación	Sintaxis	Devuelve	Expresión	Valor
Igualdad	Hora= Hora	Booleano	?01:02:03? = ?01:02:03?	True
			?01:02:03? = ?01:02:04?	False
Desigualdad	Hora# Hora	Booleano	?01:02:03? # ?01:02:04?	True
			?01:02:03? # ?01:02:03?	False
Mayor que	Hora> Hora	Booleano	?01:02:04? > ?01:02:03?	True
			?01:02:03? > ?01:02:03?	False
Menor que	Hora< Hora	Booleano	?01:02:03? < ?01:02:04?	True
			?01:02:03? < ?01:02:03?	False
Mayor que o igual a	Hora>= Hora	Booleano	?01:02:03? >=?01:02:03?	True
			?01:02:03? >=?01:02:04?	False
Menor que o igual a	Hora<= Hora	Booleano	?01:02:03? <=?01:02:03?	True
			?01:02:04? <=?01:02:03?	False

Comparaciones de punteros

Con:

```
` vPtrA y vPtrB apuntan al mismo objeto
vPtrA:=->unObjeto
vPtrB:=->unObjeto
` vPtrC apunta a otro objeto
vPtrC:=->otroObjeto
```

Operación	Sintaxis	Devuelve	Expresión	Valor
Igualdad	Puntero= Puntero	Booleano	vPtrA = vPtrB	True
			vPtrA = vPtrC	False
Desigualdad	Puntero# Puntero	Booleano	vPtrA # vPtrC	True
			vPtrA # vPtrB	False

Más sobre comparaciones de cadenas

- Las cadenas se comparan carácter por carácter (excepto en el caso de búsquedas por palabras claves, ver a continuación).
- Cuando se comparan cadenas, no se tienen en cuenta las mayúsculas y minúsculas; por lo tanto, "a"="A" devuelve TRUE. Para saber si los caracteres están en mayúsculas o minúsculas, compare sus códigos de caracteres. Por ejemplo, la siguiente expresión devuelve FALSE:

```
Character code("A")=Character code("a") // porque 65 no es igual a 97
```

- Cuando se comparan cadenas, los caracteres diacríticos se comparan utilizando la tabla de comparación de caracteres del sistema de su ordenador. Por ejemplo, las siguientes expresiones devuelven TRUE:

```
"n"="ñ"
"n"="Ñ"
"A"="å"
 // etc.
```

- A diferencia de otras comparaciones de cadena, las búsquedas por palabras claves buscan "palabras" en "textos": las palabras se evalúan global e individualmente. El operador % siempre devuelve False si la búsqueda concierne varias palabras o sólo parte de una palabra (por ejemplo, una silaba). Las "palabras" son cadenas de caracteres rodeadas por "separadores," que son los espacios y los caracteres de puntuación. Un apostrofe, como en "Today's", por lo general se considera como parte de la palabra, pero se ignorará en ciertos casos (ver las reglas abajo). Los números pueden buscarse

porque son evaluados como cadenas; sin embargo, los separadores decimales (. ,) y otros símbolos (moneda, temperatura, etc.) se ignorarán.

```
"Alpha Bravo Charlie"%"Bravo" // Devuelve True
"Alpha Bravo Charlie"%"vo" ` Devuelve False
"Alpha Bravo Charlie"%"Alpha Bravo" // Devuelve False
"Alpha,Bravo,Charlie"%"Alpha" // Devuelve True
"Software and Computers"%"comput@" // Devuelve True
```

***Notas**:*
- 4D utiliza la librería ICU para la detección de palabras claves. Para mayor información sobre las reglas implementadas, por favor visite la siguiente dirección:http://www.unicode.org/unicode/reports/tr29/#Word_Boundaries
*- En la versión japonesa, 4D utiliza por defecto la librería Mecab en lugar de ICU, para la detección de palabras claves. Para mayor información, consulte **Soporte de Mecab (versión japonesa)**.*

- *El carácter arroba (@) puede utilizarse en toda comparación de cadenas en reemplazo de uno o varios caracteres. Por ejemplo, la siguiente expresión es TRUE:*

```
"abcdefghij"="abc@"
```

El carácter arroba debe utilizarse en el segundo operando (la cadena a la derecha del operador) para igualar todo número de caracteres. La siguiente expresión es FALSE, porque la arroba @ se considera como un solo carácter en el primer operando:

```
"abc@"="abcdefghij"
```

La arroba significa "uno o más caracteres o nada". Las siguientes expresiones son TRUE:

```
"abcdefghij"="abcdefghij@"
"abcdefghij"="@abcdefghij"
"abcdefghij"="abcd@efghij"
"abcdefghij"="@abcdefghij@"
"abcdefghij"="@abcde@fghij@"
```

Por otra parte, en cualquier caso, una comparación de cadenas con dos arrobas consecutivas siempre devolverá FALSE. La siguiente expresión es FALSE:

```
"abcdefghij"="abc@@fg"
```

Cuando el operador de comparación es o contiene un símbolo < o >, sólo soporta la comparación con un solo carácter comodín al final del operando:

```
"abcd"<="abc@" // Comparación válida
"abcd"<="abc@ef"/Comparación no válida/
```

***Consejo:** si quiere ejecutar comparaciones o búsquedas utilizando @ como un carácter (y no como un comodín), tiene dos opciones:*

- *Utilice la instrucción **Character code** **(At sign)**.*
  Imagine, por ejemplo, que quiere saber si una cadena termina con el carácter @.
  - la siguiente expresión (si $vsValor no está vacía) siempre es TRUE:

```
($vsValue[[Length($vsValue)]]="@")
```

- la siguiente expresión se evaluará correctamente:

```
(Character code($vsValue[[Length($vsValue)]])#64)
```

- *Utilice la opción "Considerar @ como un comodín sólo al comienzo o al final de patrones de texto", accesible utilizando la caja de diálogo de Preferencias.*
  Esta opción le permite definir cómo se interpreta el carácter @ cuando se incluye en una cadena de caracteres. Como tal, puede influir en cómo los operadores de comparación se utilizan en Query u Order By. Para mayor información, consulte el Manual de Diseño.

Operadores de fechas

Una expresión que utiliza un operador de fechas devuelve una fecha o un número, dependiendo de la operación. Todas las operaciones de fechas devuelven valores exactos, teniendo en cuenta los cambios de años y los años bisiestos. La siguiente tabla muestra los operadores de fechas:

Operación	***Sintaxis***	***Devuelve***	***Expresión***	***Valor***
Diferencia de fechas	*Fecha – Fecha*	*Número*	*!1997-01-20! - !1997-01-01!*	*19*
Adición de días	*Fecha + Número*	*Fecha*	*!1997-01-20! + 9*	*!1997-01-29!*
Sustracción de días	*Fecha – Número*	*Fecha*	*!1997-01-20! - 9*	*!1997-01-11!*

Operadores lógicos

4D soporta dos operadores lógicos que trabajan sobre expresiones booleanas: conjunción (AND) y disyunción incluyente (OR). Un AND lógico devuelve TRUE si ambas expresiones son TRUE. Un OR lógico devuelve TRUE si al menos una de las expresiones es TRUE.

*4D también ofrece funciones booleanas **True**, **False**, y **Not**. Para mayor información, consulte la descripción de estos comandos.*

La siguiente tabla describe los operadores lógicos:

Operación	***Sintaxis***	***Devuelve***	***Expresión***	***Valor***
AND	*Booleano y Booleano*	*Booleano*	*("A" = "A") & (15 # 3)*	*True*
			("A" = "B") & (15 # 3)	*False*
			("A" = "B") & (15 = 3)	*False*
OR	*Booleano | Booleano*	*Booleano*	*("A" = "A") | (15 # 3)*	*True*
			("A" = "B") | (15 # 3)	*True*
			("A" = "B") | (15 = 3)	*False*

La siguiente es la tabla de verdad para el operador lógico AND:

Expr1	***Expr2***	***Expr1 & Expr2***
True	*True*	*True*
True	*False*	*False*
False	*True*	*False*
False	*False*	*False*

La siguiente es la tabla de verdad para el operador lógico OR:

Expr1	***Expr2***	***Expr1 | Expr2***
True	*True*	*True*
True	*False*	*True*
False	*True*	*True*
False	*False*	*False*

Operadores numéricos

Una expresión que utiliza un operador numérico devuelve un número. La siguiente tabla muestra los operadores numéricos:

Operación	*Sintaxis*	*Devuelve*	*Expresión*	*Valor*
Adición	*Número + Número*	*Número*	*2 + 3*	*5*
Sustracción	*Número – Número*	*Número*	*3 – 2*	*1*
Multiplicación	*Número * Número*	*Número*	*5 * 2*	*10*
División	*Número /Número*	*Número*	*5 / 2*	*2.5*
División entera	*Número \ Número*	*Número*	*5 \ 2*	*2*
Módulo	*Número % Número*	*Número*	*5 % 2*	*1*
Exponenciación	*Número ^ Número*	*Número*	*2 ^ 3*	*8*

El operador módulo % divide el primer número por el segundo y devuelve el resto de la división entera. Estos son algunos ejemplos:

- *10 % 2 devuelve 0 porque 10 divido 2 no da resto.*
- *10 % 3 devuelve 1 porque el resto es 1.*
- *10.5 % 2 devuelve 0 porque el resto no es un número entero.*

***Advertencia**:*

- *El operador módulo % devuelve valores significativos con números que están en la categoría de Entero largo (de menos 2^31 a 2^31 menos 1). Para calcular el módulo con números fuera de este rango, utilice el comando **Mod**.*
- *El operador división entera \ devuelve valores significativos de números enteros únicamente.*

Operadores de imágenes

Una expresión que utiliza un operador de imágenes devuelve una imagen. La siguiente tabla muestra los operadores de imágenes.

Operación	Sintaxis	Acción
Concatenación horizontal	Imag1 + Imag2	Añade Imag2 a la derecha de Imag1
Concatenación Vertical	Imag1 / Imag2	Añade Imag2 debajo de Imag1
Superposición exclusiva (*)	Imag1 & Imag2	Superpone Imag2 sobre Imag1 (Imag2 al fondo)
Superposición inclusiva (*)	Imag1 | Imag2	Superpone Imag2 a Imag1 y devuelve la máscara resultante si las dos imágenes son del mismo tamaño
Desplazamiento Horizontal	Imagen + Número	Mueve la imagen horizontalmente un número de píxeles
Desplazamiento Vertical	Imagen / Número	Mueve la imagen verticalmente un número de píxeles
Redimensionamiento	Imagen * Número	Redimensiona Imagen en el porcentaje Número
Extensión horizontal	Imagen *+ Número	Redimensiona Imagen horizontalmente en el porcentaje Número
Extensión vertical	Imagen */ Número	Redimensiona Imagen verticalmente en el porcentaje Número

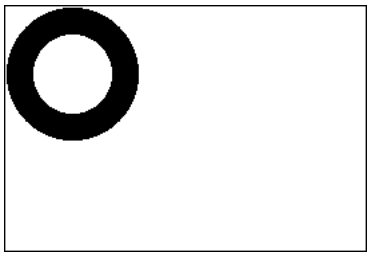
Los dos operadores & y | siempre devuelven una imagen tipo mapa de bits, sin importar la naturaleza de las dos imágenes fuente. La razón es que 4D primero dibuja las imágenes en memoria en mapas de bits, luego calcula la imagen resultante aplicando el operador OR en los píxeles del mapa de bits.

Los otros operadores de imágenes devuelven imágenes vectoriales si las dos imágenes fuente son vectoriales. Recuerde, sin embargo, que las imágenes impresas con el formato de salida ***On Background*** *se imprimen como mapas de bits.*

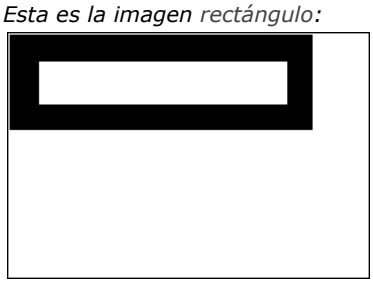
Ejemplo

En los siguientes ejemplos, todas las imágenes se muestran utilizando el formato de salida ***On Background****.*

Esta es la imagen círculo:



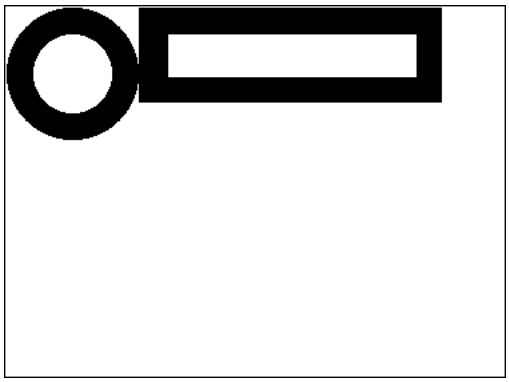
Esta es la imagen rectángulo:



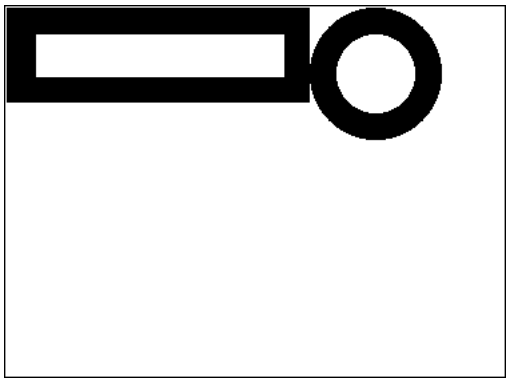
En los siguientes ejemplos, cada expresión es seguida por su representación gráfica.

- *Concatenación horizontal*

circulo+rectangulo ` Situa el rectángulo a la derecha del círculo



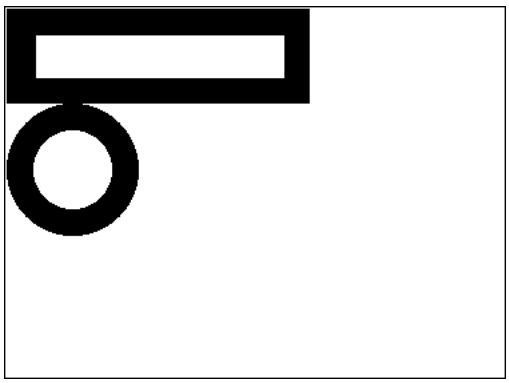
rectángulo+círculo ` Situa el círculo a la derecha del rectángulo



- *Concatenación vertical*

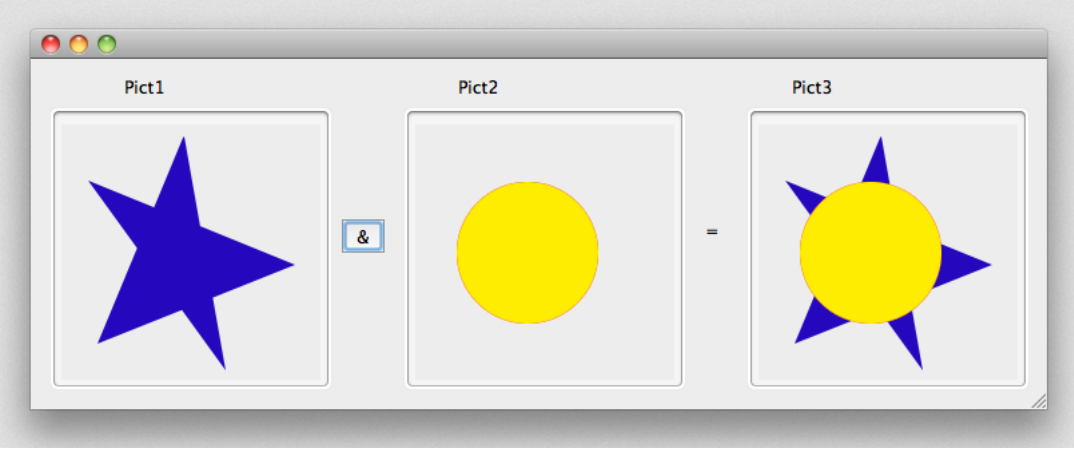
circulo/rectángulo ` Poner el rectángulo bajo el círculo

rectángulo/círculo ` Pone el círculo bajo el rectángulo



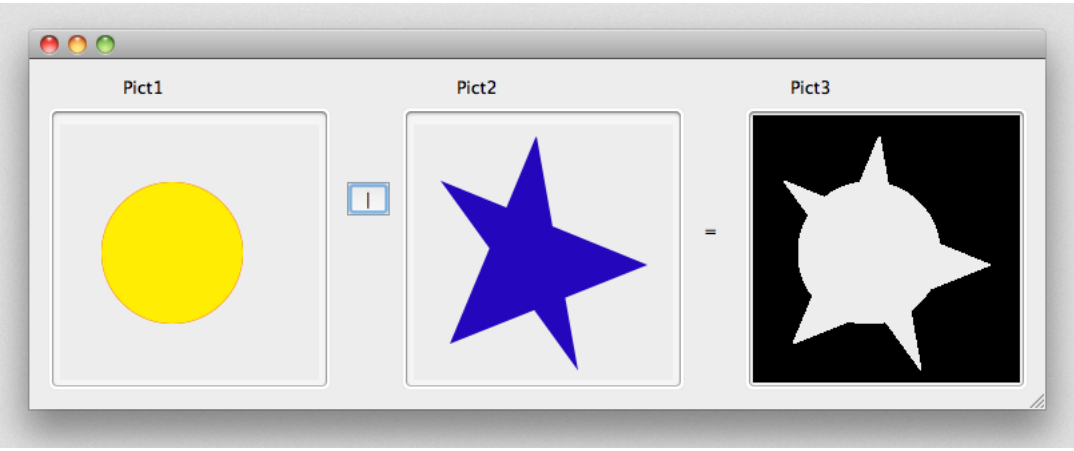
- *Superposición exclusiva*

```
Pict3:=Pict1 & Pict2 // Superimpone Imag2 sobre Imag1
```



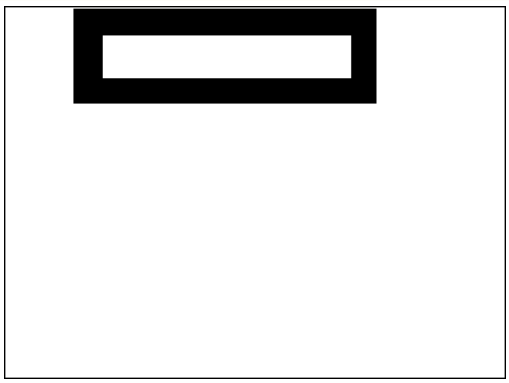
- *Superposición inclusiva*

Pict3:=Pict1|Pict2 // Recupera la máscara resultante de la superposición de dos imágenes del mismo tamaño

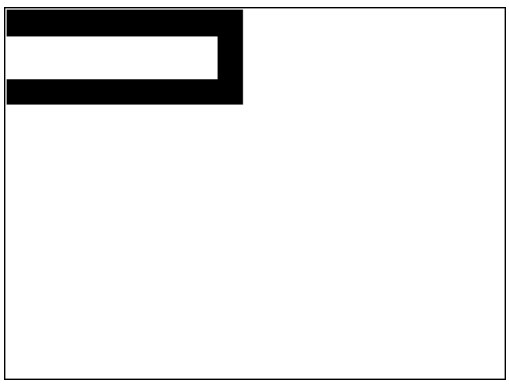


- *Desplazamiento horizontal*

rectángulo+50 ` Mueve el rectángulo 50 píxeles a la derecha

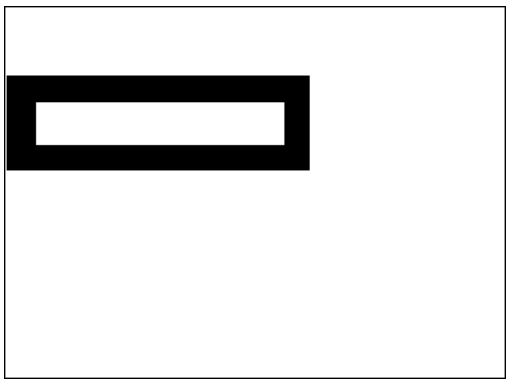


rectángulo-50 ` Mueve el rectángulo 50 píxeles a la izquierda

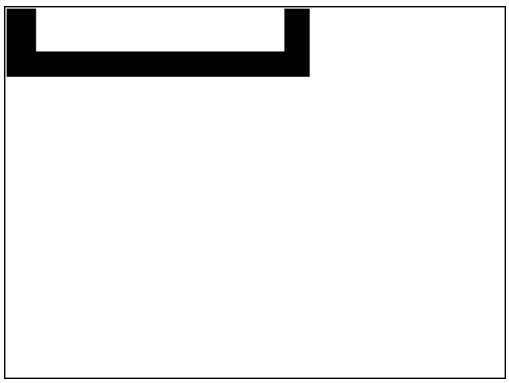


- *Desplazamiento vertical*

rectángulo/50 ` Mueve el rectángulo 50 píxeles hacia abajo



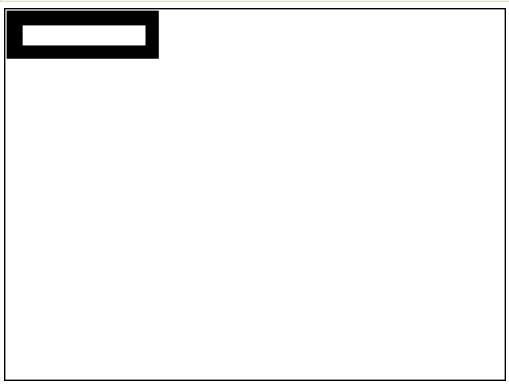
rectángulo/-20 ` Mueve el rectángulo 20 píxeles hacia arriba



- *Redimensionamiento*

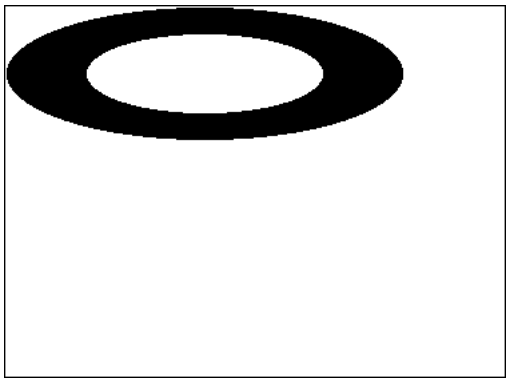
*rectángulo*1.5* ` Aumenta el tamaño del rectángulo en 50%

*rectángulo*0.5* ` El rectángulo se vuelve 50% más pequeño



- *Extensión horizontal*

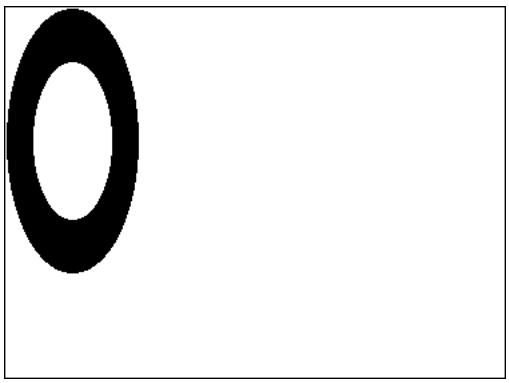
círculo+3* ` El círculo se vuelve 3 veces más grande



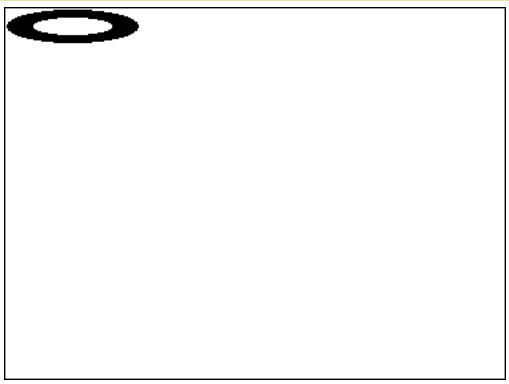
círculo+0.25* ` El ancho del círculo se reduce un cuarto de su tamaño original

- *Extensión vertical*

círculo/2* ` El círculo dobla su altura



círculo/0.25* ` La altura del círculo se reduce un cuarto de su tamaño original



Operadores de cadenas

Una expresión que utiliza un operador de cadenas devuelve una cadena. La siguiente tabla muestra los operadores de cadena:

Operación	***Sintaxis***	***Devuelve***	***Expresión***	***Valor***
Concatenación	*Cadena+Cadena*	*Cadena*	*"abc" + "def"*	*"abcdef"*
Repetición	*Cadena* Número*	*Cadena*	*"ab" * 3*	*"ababab"*

Operadores de horas

Una expresión que utiliza un operador de horas devuelve una hora o un número, dependiendo de la operación. La siguiente tabla muestra los operadores de horas:

Operación	Sintaxis	Devuelve	Expresión	Valor
Adición	Hora+ Hora	Hora	?02:03:04? + ?01:02:03?	?03:05:07?
Substracción	Hora– Hora	Hora	?02:03:04? – ?01:02:03?	?01:01:01?
Adición	Hora + Número	Número	?02:03:04? + 65	7449
Substracción	Hora – Número	Número	?02:03:04? – 65	7319
Multiplicación	Hora * Número	Número	?02:03:04? * 2	14768
División	Hora / Número	Número	?02:03:04? / 2	3692
División entera	Hora \ Número	Número	?02:03:04? \ 2	3692
Módulo	Hora % Hora	Hora	?20:10:00? % ?04:20:00?	?02:50:00?
Módulo	Hora % Número	Número	?02:03:04? % 2	0

Ejemplo 1

*Puede combinar expresiones de tipo hora y de tipo numérico utilizando los comandos **Time** y **Time string**. Por ejemplo:*

```
  //La siguiente línea asigna a la variable $vlSegundos el número de segundos que transcurrirán entre la media noche y una hora a partir de ahora
$vlSeconds:=Current time+3600
  //La línea siguiente asigna a $vHSoon la hora que será en una hora
$vhSoon:=Time(Current time+3600)
```

La segunda línea puede escribirse de una manera más simple:

```
  //La línea siguiente asigna a $vhSoon la hora que será en una hora
$vhSoon:=Current time+?01:00:00?
```

Ejemplo 2

Algunas situaciones podrían requerir que usted convierta una expresión de tiempo en una expresión numérica.
*Por ejemplo, usted abre un documento utilizando **Open document**, el cual devuelve un número de referencia de documento (**DocRef**) que es una expresión de tipo hora. Más adelante, usted quiere pasar **DocRef** a una rutina de plug-in 4D que espera un valor numérico como número de referencia de documento. En tal caso, utilice la adición con 0 (cero) para obtener un valor numérico de un valor de tiempo, pero sin cambiar su valor. Por ejemplo:*

```
  ` Seleccionar y abrir un documento
$vhDocRef:=Open document("")
If(OK=1)
  ` Pase la expresión hora DocRef como una expresión numérica para una rutina de extensión 4D
  DO SOMETHING SPECIAL(0+$vhDocRef)
End if
```

Ejemplo 3

El operador Modulo permite añadir horas teniendo en cuenta el formato 24 horas de un día:

```
$t1:=?23:00:00?  // Son las 23:00 p.m.
 // Queremos añadir 2 horas y media
$t2:=$t1 +?02:30:00?  // Con una adición simple, $t2 es ?25:30:00?
$t2:=($t1 +?02:30:00?)%?24:00:00?  // $t2 es ?01:30:00? y son la 1:30 a.m. del siguiente día
```

ORDA - DataClass

---

*Para saber cómo manejar las dataclasses en su código, consulte la página **Clases de datos** en la Guía del desarrollador 4D.*

- *dataClass.{attributeName}* New 17.0
- *dataClass.all* New 17.0
- *dataClass.fromCollection* New 17.0
- *dataClass.get* New 17.0
- *dataClass.new* New 17.0
- *dataClass.newSelection* New 17.0
- *dataClass.query* New 17.0

dataClass.{attributeName}

Parámetro	Tipo	Descripción
dataClass.{attributeName}	*DataClassAttribute*	*Dataclass attribute description*

Descripción

The attributes of dataclasses are objects that are available directly as properties of these classes.
The returned objects are of the DataClassAttribute type. These objects have properties that you can use and read to get information about your dataclass attributes. These properties are listed in the ***ORDA - DataClassAttribute*** *section.*
Note: *Dataclass attributes can also be reached using the alternate bracket syntax [ ] (see example).*

Ejemplo

```
$salary:=ds.Employee.salary //returns the salary attribute in the Employee dataclass
$compCity:=ds.Company["city"] //returns the city attribute in the Company dataclass
```

dataClass.all( )

dataClass.all () -> Resultado

Parámetro	Tipo		Descripción
Resultado	EntitySelection		Referencias sobre todas las entidades relacionadas con la Dataclass

Descripción

*El método **dataClass.all( )** consulta el datastore para buscar todas las entidades relacionadas con la dataclass y las devuelve como una entity selection.*

Las entidades se devuelven en el orden por defecto, que es inicialmente el orden en el que se crearon. Sin embargo, tenga en cuenta que, si se han eliminado entidades y se han agregado nuevas, el orden predeterminado ya no refleja el orden de creación.

Si no se encuentra una entidad correspondiente, se devuelve una selección de entidad vacía.

Se aplica carga diferida.

Ejemplo

```
C_OBJECT($allEmp)
$allEmp:=ds.Employee.all()
```

dataClass.fromCollection( )

dataClass.fromCollection (colObjeto) -> Resultado

Parámetro	Tipo		Descripción
colObjeto	Collection	➡	Colección de objetos a mapear con entidades
Resultado	EntitySelection	➲	Entity selection llenada desde la colección

Descripción

*El método **dataClass.fromCollection( )** modifica o crea entidades en la dataclass utilizando la colección de objetos colObjeto, y devuelve la selección de entidades correspondiente.*

*En el parámetro colObjeto, pase una colección de objetos para crear o modificar las entidades de la dataclass. Los nombres de las propiedades deben corresponder a los de la dataclass. Si un nombre de propiedad no existe en la dataclass, se ignora. Si un valor de atributo no está definido en la colección para una entidad creada, el atributo toma el valor **null**.*
*La correspondencia entre los objetos de la colección y las entidades se hace a nivel de los **nombres de atributos** y de su **tipo de datos**. Si una propiedad de objeto tiene el mismo nombre que un atributo de entidad pero sus tipos no son compatibles, el atributo de la entidad no recibe valor.*

Modo creación o modificación
Para cada objeto de colObjeto:

- *Si el objeto contiene una propiedad booleana "__NEW" definida en **false** (o no contiene una propiedad "__NEW"), la entidad se modifica o crea con los valores correspondientes de las propiedades del objeto. No se realiza ninguna verificación con respecto a la llave primaria:*
  - *Si la llave primaria se da en el objeto y existe, la entidad se modifica.*
    En este caso, la llave primaria puede darse tal cual o vía la propiedad "__KEY" (que contiene el valor de la propiedad primaria).
  - *Si la llave primaria se da (tal cual) y no existe, se crea la entidad*
  - *Si la llave primaria no se da, la entidad se crea y la llave primaria se asigna según las reglas vigentes de la base de datos.*

- *Si el objeto contiene una propiedad booleana "__NEW" definida en **true**, la entidad se crea con los valores correspondientes de las propiedades del objeto. Se realiza una verificación con respecto a la llave primaria:*
  - *Si la llave primaria se da (tal cual) y existe, se genera un error*
  - *Si la llave primaria se da (tal cual) y no existe, se crea la entidad*
  - *Si la llave primaria no se da en el objeto, la entidad se crea y la llave primaria es asignada según las reglas vigentes de la base de datos.*
    *Cuando la propiedad "__NEW" de establece en **true**, la llave primaria se debe dar como está (no con la propiedad __KEY).*

***Nota:** la propiedad "__KEY" que contiene un valor se tiene en cuenta solo cuando la propiedad "__NEW" se define en **false** (o se omite) y existe una entidad correspondiente. En todos los demás casos, el valor de la propiedad "__KEY" se ignora, el valor de la llave primaria se debe pasar "tal cual".*

Entidades relativas
Los objetos de colObjeto pueden contener uno o más objetos anidados que presentan una o más entidades relativas, que pueden ser útiles para crear o actualizar relaciones entre entidades.
Los objetos anidados que presentan entidades relacionadas deben contener el atributo ___KEY (llenado con la llave primaria de la entidad relativa) o la llave primaria de la entidad relativa. El uso del atributo __KEY permite la independencia del nombre de la llave primaria.

***Nota**: el contenido de las entidades relacionadas no se puede crear/actualizar a través de este mecanismo.*

Stamp
Si una propiedad __STAMP es dada, se realiza una verificación con el stamp (marcador interno) de la entidad en el datastore y se puede devolver un error ("El stamp no coincide con el actual para el registro# XX de la tabla XXXX").

Ejemplo 1

Queremos actualizar una entity existent. La propiedd __NEW no es dada, la llave primaria employee es dada y existe:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.ID:=668 //Llave primaria existente en la tabla Employee
$emp.firstName:="Arthur"
$emp.lastName:="Martin"
$emp.employer:=New object("ID";121) //Llave primaria existente en dataClass relativa Company
 // Para este empleado, podemos cambiar Company utilizando otro PK existente en la dataClass relativa Company
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

Ejemplo 2

Queremos actualizar una entidad existente. La propiedad __NEW no se da, la llave primaria del empleado está con el atributo __KEY y existe:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.__KEY:=1720  //PK existente en tabla Employee
$emp.firstName:="John"
$emp.lastName:="Boorman"
$emp.employer:=New object("ID";121)  //PK existente en la dataClass Company relacionada
 // Para este empleado, podemos cambiar Company utilizando otro PK existente en la dataClass Company relacionada
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

Ejemplo 3

Queremos crear una nueva entidad a partir de una colección:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.firstName:="Victor"
$emp.lastName:="Hugo"
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

Ejemplo 4

Queremos crear una entidad. La propiedad __NEW es True, la llave primaria del empleado no se da:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.firstName:="Mary"
$emp.lastName:="Smith"
$emp.employer:=New object("__KEY";121)  //Llave primaria existente en la dataClass relativa Company
$emp.__NEW:=True
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

Ejemplo 5

Queremos crear una entidad. La propiedad __NEW se omite, la llave primaria employee es dada y no existe:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.ID:=10000  //Llave primaria inexistente
$emp.firstName:="Françoise"
$emp.lastName:="Sagan"
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

Ejemplo 6

En este ejemplo, la primera entidad será creada y guardada pero la segunda fallará ya que ambas utilizan la misma llave primaria:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$emp2;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.ID:=10001  // Llave primaria inexistente
$emp.firstName:="Simone"
$emp.lastName:="Martin"
$emp.__NEW:=True
$empsCollection.push($emp)

$emp2:=New object
$emp2.ID:=10001  // Misma llave primaria, ya existente
$emp2.firstName:="Marc"
$emp2.lastName:="Smith"
$emp2.__NEW:=True
$empsCollection.push($emp2)
$employees:=ds.Employee.fromCollection($empsCollection)
 //first entity is created
 //Error llave duplicada para la segunda entidad
```

dataClass.get( )

dataClass.get (llavePrimaria) -> Resultado

Parámetro	*Tipo*		*Descripción*
llavePrimaria	*Entero largo, Texto*	→	*Valor de llave primaria de la entidad a recuperar*
Resultado	*Entity*	↩	*Entidad que coincide con la llave primaria designada*

Descripción

*El método **dataClass.get( )** busca la dataclass para recuperar la entidad que coincide con el parámetro llavePrimaria.*

*En llavePrimaria, pase el valor de la llave primaria de la entidad a recuperar. El tipo de valor debe coincidir con el tipo de llave primaria establecida en el datastore (entero largo o texto). También puede asegurarse de que el valor de la llave primaria siempre se devuelva como texto utilizando el método **entity.getKey( )** con el parámetro dk key as string.*

*Si no se encuentra ninguna entidad con llavePrimaria, se devuelve una entidad **Null**.*

Se aplica carga diferida, lo que significa que los datos relacionados se cargan desde el disco solo cuando es necesario.

Ejemplo

```
C_OBJECT($entity)
$entity:=ds.Employee.get(167) // devuelve la entidad cuyo valor de llave primario es 167
$entity:=ds.Invoice.get("DGGX20030") // devuelve la entidad cuyo valor de llave primaria es "DGGX20030"
```

dataClass.new( )

dataClass.new () -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entity*		*Nueva entidad correspondiente al dataClass*

Descripción

*El método **dataClass.new( )** crea en la memoria y devuelve una nueva entidad en blanco relacionada con la dataClass.*
*El objeto entidad se crea en la memoria y no se guarda en la base de datos hasta que se llame al método **entity.save( )**. Si la entidad se borra antes de guardarse, no se puede recuperar.*

Ejemplo

Este ejemplo crea una nueva entidad en la dataClass "Log" y registra información en el atributo info:

```
C_OBJECT($entity)
$entity:=ds.Log.new() //crea una referencia
$entity.info:="New entry" //almacena información
$entity.save() //guarda la entidad
```

dataClass.newSelection( )

dataClass.newSelection ({mantenerOrden}) -> Resultado

Parámetro	Tipo		Descripción
mantenerOrden	Entero largo	➡	dk keep ordered: crea una entity selection ordenada, dk non ordered or omitted: crea una entity selection no ordenada
Resultado	EntitySelection	➲	Nueva entity selection vacía relacionada con la dataclass

Descripción

*El método **dataClass.newSelection( )** crea una nueva entity selection en blanco en la memoria, relacionada con la dataclass.*

*Si desea crear una entity selection ordenada, pase el selector dk keep ordered en el parámetro mantenerOrden. Por defecto, si omite este parámetro, o si pasa el selector dk non ordered, el método crea una entity selection desordenada. Las entity selections desordenadas son más rápidas, pero no puede confiar en las posiciones de las entidades. Para más información, consulte el párrafo **Ordenadas vs No ordenadas** en la Guía del desarrollador 4D.*

*Cuando se crea, la entity selection no contiene ninguna entidad (mySelection.length devuelve 0). Este método le permite crear entity selections gradualmente realizando llamadas posteriores al método **add( )**.*

Ejemplo

```
C_OBJECT($USelection;$OSelection)
$USelection:=ds.Employee.newSelection()  //crea una entity selection vacía desordenada
$OSelection:=ds.Employee.newSelection(dk keep ordered)  //crea una entity selection vacía ordenada
```

dataClass.query( )

dataClass.query (cadenaBusq {; valor}{; valor2 ; ... ; valorN}{; confBusq}) -> Resultado

Parámetro	Tipo		Descripción
cadenaBusq	Texto	➡	Criterio de búsqueda
valor		➡	Valor(es) a comparar al utilizar separadores
confBusq	Objeto	➡	Opciones de búsqueda: parámetros, queryPath, queryPlan
Resultado	EntitySelection	⮌	Nueva selección de entidad formada por entidades de dataClass que cumplen los criterios de búsqueda especificados en cadenaBusq

Descripción

El método ***dataClass.query( )*** *busca entidades que cumplan los criterios de búsqueda especificados en cadenaBusq y (opcionalmente) valor, para todas las entidades de la clase de datos o de la selección de entidades y devuelve un nuevo objeto de tipo EntitySelection que contiene todas las entidades de la clase de datos que fueron encontradas. Se aplica carga diferida.*

Si no se encuentran entidades coincidentes, se devuelve una EntitySelection vacía.

Parámetro cadenaBusq

El parámetro cadenaBusq utiliza la siguiente sintaxis:

```
rutaAtributo comparador valor {operadorLogico rutaAtributo comparador valor}
```

donde:

- ***rutaAtributo*** *: nombre del atributo de clase de datos en el que desea ejecutar la búsqueda, por ejemplo, "country". Este parámetro puede ser cualquier ruta de atributo válida, como "country.name".*
  En el caso de una ruta de atributo cuyo tipo es Collection, la notación [] se usa para manejar todas las ocurrencias.

- ***comparador****: símbolo que compara rutaAtributo y valor. Los siguientes símbolos son soportados:*

Comparación	Símbolo(s)	Comentario
Igual a	=, ==	Obtiene datos coincidentes, soporta el comodín (@), no distingue entre mayúsculas y minúsculas ni diacrítico.
	===, IS	Obtiene datos que coinciden, considera el comodín (@) como un carácter estándar, no distingue entre mayúsculas y minúsculas ni es diacrítico
No igual a	#, !=	Soporta el comodín (@)
	!==, IS NOT	Considera el comodín (@) como un carácter estándar,
	#, !=, is not	
Menor que	<	
Mayor que	>	
Menor o igual a	<=	
Mayor que o igual a	>=	
Incluido en	IN	Obtiene datos iguales para al menos uno de los valores en una colección o en un conjunto de valores
Condición No aplicada a una instrucción	NOT	Los paréntesis son obligatorios cuando se usa Not antes de una instrucción que contiene varios operadores
Contiene palabra clave	%	Las palabras claves se pueden usar en atributos de tipo texto o imagen

- ***valor****: el valor a comparar con el valor actual de la propiedad de cada objeto en la colección o selección de entidad. Puede ser cualquier expresión del mismo tipo de datos que la propiedad o un separador de posición: paramIndex (ver abajo).*
  Los valores constantes se dan entre comillas simples. Las siguientes palabras claves están prohibidas para las constantes: true, false. Puede comparar el valor ***Null*** *en una búsqueda utilizando la palabra clave "null". Esta búsqueda encontrará propiedades null y undefined.*
  Para buscar una cadena dentro de una cadena (una búsqueda "contiene"), utilice el símbolo comodín (@) en valor para aislar la cadena que se va a buscar como se muestra en este ejemplo: "@Smith@".
  Para valores numéricos, los separadores decimales son el punto. Las fechas deben ser provistas con el formato "AAAA-MM-DD".
  En el caso de una búsqueda con un comparador IN, valor debe ser una colección, o valores que coincidan con el tipo de la ruta del atributo entre [] separados por comas (para cadenas, "los caracteres se deben escapar con "\").

- ***operadorLogico****: utilizado para unir múltiples condiciones en la búsqueda (opcional). Puede usar uno de los siguientes operadores lógicos (puede pasar el nombre o el símbolo):*

Conjunción	Símbolo(s)
AND	&, &&, and
OR	|, ||, or

Utilizando comillas

Cuando utiliza comillas dentro de búsquedas, debe usar comillas simples ' ' dentro de la búsqueda y comillas dobles " " para abarcar toda la búsqueda, de lo contrario, se devuelve un error. Por ejemplo:

```
"employee.name = 'smith' AND employee.firstname = 'john'"
```

Utilizando paréntesis
Puede utilizar paréntesis en búsquedas para dar prioridad al cálculo. Por ejemplo, puede organizar una búsqueda de la siguiente manera:

```
"(employee.age >= 30 OR employee.age <= 65) AND (employee.salary <= 10000 OR employee.status = 'Manager')"
```

Parámetro valor (y marcadores de posición)

Los parámetros valor se deben usar cuando la consulta se genera con marcadores de posición. Los marcadores de posición son etiquetas que se inserta en cadenas de búsqueda y que se reemplazan por otro valor cuando se evalúa la cadena de búsqueda. Puede usar hasta 128 parámetros de valor.

***Nota:** los valores para los marcadores de posición también se pueden pasar como una colección en la propiedad de parámetros del parámetro opcional confBusq (solo para las consultas entitySelection y dataClass). Para más información, consulte el párrafo **Parámetro confBusq**.*

En cadenaBusq, inserte :paramIndex para cada marcador de posición (lo que significa "use el parámetro paramIndex de la consulta como el valor a comparar") y luego, pase el(los) valor(es) solicitado(s) como parámetro(s) valor. Por ejemplo, para consultar a los empleados que viven en Chicago y ganan menos de 10 000, puede escribir:

```
"employee.city = :1 & employee.salary < :2"; "Chicago";10000
```

El valor se evalúa una vez al comienzo de la consulta; no se evalúa para cada elemento.

***Se recomienda utilizar marcadores de posición en consultas** por dos motivos:*

1. ***Evita la inserción de código malicioso**: si utiliza directamente variables llenadas por usuarios dentro de la cadena de consulta, un usuario podría modificar las condiciones de consulta al ingresar argumentos de consulta adicionales. Por ejemplo, imagine una cadena de consulta como:*

   ```
   $vquery:="status = 'public'  &  name = "+myname //user enters their name
   $result:=$col.query($vquery)
   ```

   Esta consulta parece segura ya que los datos no públicos se filtran. Sin embargo, si el usuario introduce en el área $myname algo así como smith OR status='private', la cadena de consulta se modificaría en el paso de interpretación y podría devolver datos privados.
   Al utilizar marcadores de posición, no es posible prevalecer sobre las condiciones de seguridad:

   ```
   $result:=$col.query("status='public'  &  name=:1";$myvar)
   ```

   En este caso, si el usuario introduce OR status='private' en el área $myvar, no se interpretará en la cadena de consulta, sino que solo se pasará como un valor. Buscar una persona llamada "OR status='private'" simplemente fallará.
2. *Evita tener que preocuparse por problemas de formato. Además, permite el uso de variables o expresiones en argumentos de consulta, por ejemplo:*

   ```
   $result:=$col.query("address.city = :1  &  name =:2";$city;$myVar+"@")
   ```

Buscando valores null
Cuando busca valores null, no puede usar la sintaxis de marcador de posición porque el motor de consulta considera null como un valor de comparación inesperado. Por ejemplo, si ejecuta la siguiente consulta:

```
$vSingles:=ds.Person.query("spouse = :1";Null) // NO funcionará
```

No obtendrá el resultado esperado porque el valor null será evaluado por 4D como un error resultante de la evaluación del parámetro (por ejemplo, un atributo procedente de otra consulta). Para este tipo de búsquedas, debe usar la sintaxis de consulta directa:

```
$vSingles:=ds.Person.query("spouse = null") //sintaxis correcta
```

Parámetro confBusq

***Nota:** este parámetro solo es soportado por los métodos **entitySelection.query( )** y **dataClass.query( )**.*

En el parámetro confBusq, puede pasar un objeto que contenga opciones adicionales. Las siguientes propiedades son soportadas:

Propiedad	Tipo	Descripción
parámetros	Colección	Valores a comparar cuando se usan marcadores de posición en cadenaBusq (forma alternativa de pasar valores a los marcadores de posición). Si algunos valores también se han pasado directamente en parámetros valor, estos valores se anexan a la secuencia de marcador de posición.
planBusqueda	Booleano	En la colección de entidades resultante, devuelve o no devuelve la descripción detallada de la consulta justo antes de que se ejecute, es decir, la consulta planificada. La propiedad devuelta es un objeto que incluye cada consulta planeada y subconsulta (en el caso de una consulta compleja). Esta opción es útil durante la fase de desarrollo de una aplicación. Por lo general, se utiliza junto con queryPath. Predeterminado si se omite: false
rutaBusqueda	Booleano	En la colección de entidades resultante, devuelve o no devuelve la descripción detallada de la consulta tal como se realiza. La propiedad devuelta es un objeto que contiene la ruta real utilizada para la consulta (generalmente idéntica a la de queryPlan, pero puede diferir si el motor logra optimizar la consulta), así como el tiempo de procesamiento y la cantidad de registros encontrados. Esta opción es útil durante la fase de desarrollo de la aplicación. Por defecto si se omite: false

Acerca de planBusqueda y rutaBusqueda

La información registrada en planBusqueda/rutaBusqueda incluye el tipo de consulta (indexada y secuencial) y cada subconsulta necesaria junto con los operadores de conjunción. Las rutas de búsqueda también contienen la cantidad de entidades encontradas y el tiempo requerido para ejecutar cada criterio de búsqueda. Puede resultarle útil analizar esta información mientras desarrolla sus aplicaciones. En general, la descripción del plan de búsqueda y su ruta son idénticos, pero pueden diferir porque 4D puede implementar optimizaciones dinámicas cuando se ejecuta una búsqueda para mejorar el rendimiento. Por ejemplo, el motor 4D puede convertir dinámicamente una búsqueda indexada en una secuencial si estima que es más rápida. Este caso particular puede ocurrir cuando el número de entidades que se buscan es bajo.

Por ejemplo, si ejecuta la siguiente búsqueda:

```
$sel:=ds.Employee.query("salary < :1 and employer.name = :2 or employer.revenues > :3";50000;"Lima West Kilo";10000000;New object("queryPath";True;"queryPlan";True))
```

queryPlan:

```
{Or:[{And:[{item:[index : Employee.salary ] < 50000},{item:Join on Table : Company  :  Employee.employerID = Company.ID,subquery:[{item:[index : Company.name ] = Lima West Kilo}]}]},{item:Join on Table : Company  :  Employee.employerID = Company.ID,subquery:[{item:[index : Company.revenues ] > 10000000}]}]}
```

queryPath:

```
{steps:[{description:OR,time:63,recordsfounds:1388132,steps:[{description:AND,time:32,recordsfounds:131,steps:[{description:[index : Employee.salary ] < 50000,time:16,recordsfounds:728260},{description:Join on Table : Company  :  Employee.employerID = Company.ID,time:0,recordsfounds:131,steps:[{steps:[{description:[index : Company.name ] = Lima West Kilo,time:0,recordsfounds:1}]}]}]},{description:Join on Table : Company  :  Employee.employerID = Company.ID,time:31,recordsfounds:1388132,steps:[{steps:[{description:[index : Company.revenues ] > 10000000,time:0,recordsfounds:933}]}]}]}]}
```

Ejemplos

Aquí hay varios ejemplos de consultas válidas.

Consulta estándar con marcadores de posición:

```
$entitySelection:=dataClass.query("(firstName = :1 or firstName = :2) and (lastName = :3 or lastName = :4)";"D@";"R@";"S@";"K@")
```

Consulta estándar sin marcadores de posición:

```
$entitySelection :=dataClass.query("firstName = 'S@'")
```

Consulta con un dataClass relacionado:

```
$entitySelection:=dataClass.query("lastName = :1 and manager.lastName = :2";"M@";"S@")
```

Consulta con objetos queryPlan y queryPath:

```
$entitySelection:=dataClass.query("(firstName = :1 or firstName = :2) and (lastName = :3 or lastName = :4)";"D@";"R@";"S@";"K@";New object("queryPlan";True;"queryPath";True))

 //Puede obtener estas propiedades en la selección de entidad resultante
C_OBJECT($queryPlan;$queryPath)
$queryPlan:=$entitySelection.queryPlan
$queryPath:=$entitySelection.queryPath
```

Consulta con marcadores de posición y valores dados como una colección:

```
$params:=New object
$params.parameters:=New collection("D@";"R@";"S@";"K@")
$entitySelection:=dataClass.query("(firstName = :1 or firstName = :2) and (lastName = :3 or lastName = :4)";$params)
```

Consulta con una declaración NOT:

```
$entitySelection:=dataClass.query("not(firstName=Kim)")
```

Consulta con una ruta de atributo de tipo Collection:

```
$entitySelection:=dataClass.query("additionalInfo.hobbies[].name = horsebackriding")
```

Consulta con una ruta de atributo de tipo Object:

```
$entitySelection:=ds.Employee.query("extra.eyeColor = :1";"blue")
```

Consulta con una declaración IN:

```
$entitySelection:=dataClass.query("firstName in :1";New collection("Kim";"Dixie"))
```

Consulta con una instrucción NOT (IN):

```
$entitySelection:=ds.Employee.query("not (firstName in :1)";New collection("John";"Jane"))
```

Consulta con una fecha:

```
$entitySelection:=dataClass.query("birthDate > :1";"1970-01-01")
```

ORDA - DataClassAttribute

---

*For more information about dataclass attributes, please refer to the **Dataclass attributes** paragraph in the 4D Developer's Guide.*

- *dataClassAttribute.kind* New 17.0
- *dataClassAttribute.name* New 17.0
- *dataClassAttribute.relatedDataClass* New 17.0

dataClassAttribute.kind

Parámetro	Tipo	Descripción
dataClassAttribute.kind	Cadena	Tipo de atributo

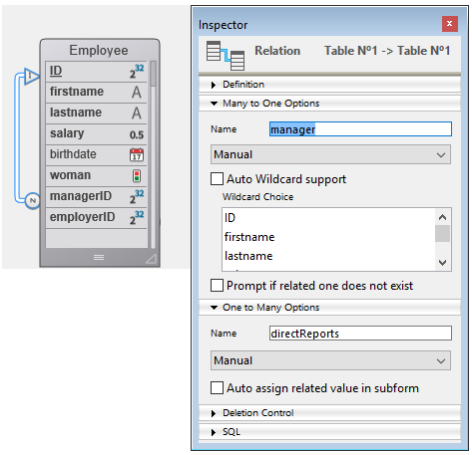
Descripción

*La propiedad **dataClassAttribute.kind** devuelve la categoría del atributo. El valor devuelto puede ser uno de los siguientes:*

- *"storage": atributo de almacenamiento (o escalar), es decir, atributo que almacena un valor, no una referencia a otro atributo*
- *"relatedEntity": N -> 1 atributo relación (referencia a una entidad)*
- *"relatedEntities": 1 -> N atributo relación N (referencia a una entity selection)*

Ejemplo

Dada la siguiente tabla y relación:




```
C_TEXT($attKind)
$attKind:=ds.Employee.lastname.kind //$attKind="storage"
$attKind:=ds.Employee.manager.kind //$attKind="relatedEntity"
$attKind:=ds.Employee.directReports.kind //$attKind="relatedEntities"
```

dataClassAttribute.name

Parámetro	Tipo	Descripción
dataClassAttribute.name	Cadena	Nombre del atributo como se define en la estructura de la base

Descripción

*La propiedad **dataClassAttribute.name** devuelve el nombre del objeto dataClassAttribute como cadena.*

Ejemplo

```
C_TEXT($attName)
$attName:=ds.Employee.lastname.name //$attName="lastname"
```

dataClassAttribute.relatedDataClass

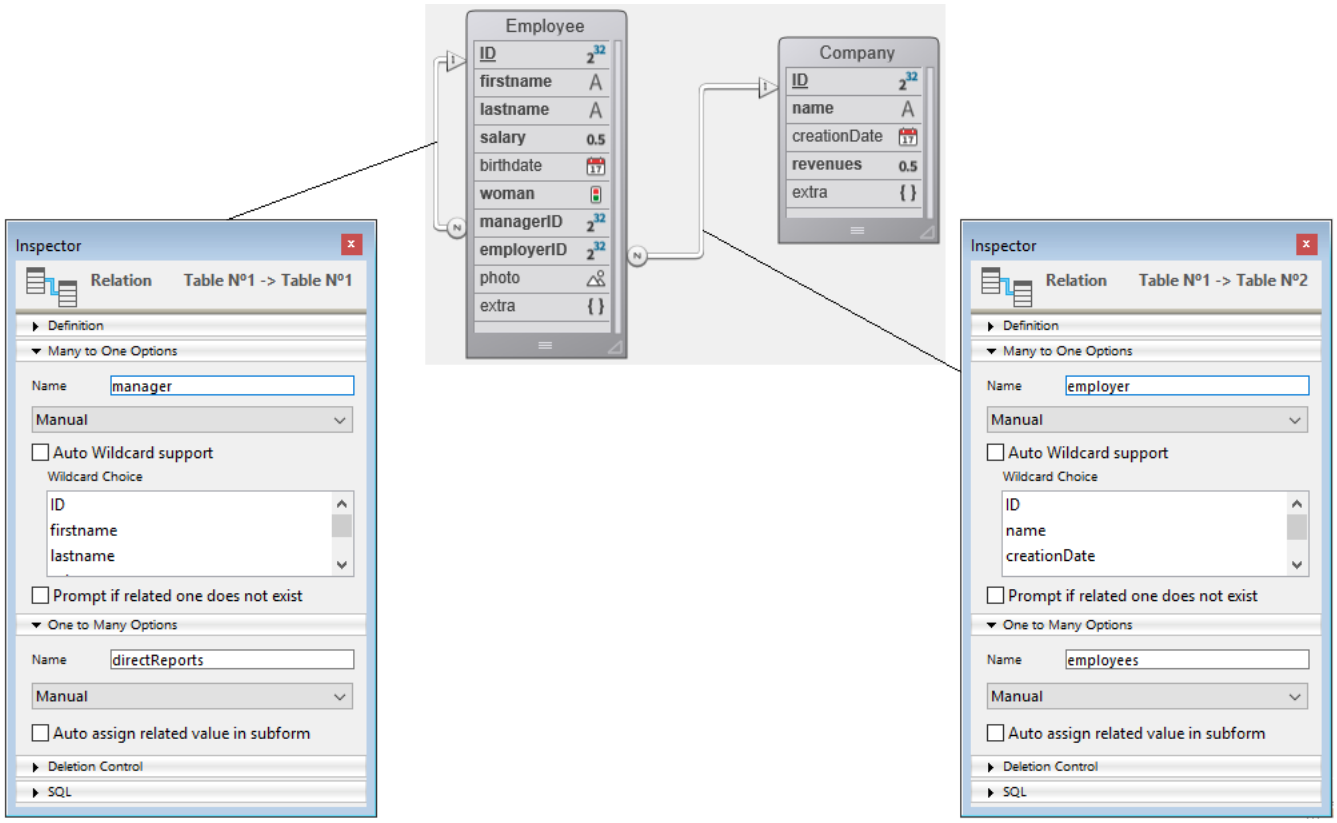
Parámetro	Tipo	Descripción
dataClassAttribute.relatedDataClass	Cadena	Nombre de la dataclass relacionada

Descripción

Nota: *esta propiedad solo está disponible con los atributos de propiedad* ***dataClassAttribute.kind*** *"relatedEntity" o "relatedEntities".*
La propiedad ***dataClassAttribute.relatedDataClass*** *devuelve el nombre de la dataclass relacionada con el atributo.*

Ejemplo

Dadas las siguientes tablas y relaciones:




```
C_TEXT($relClass1;$relClassN)
$relClass1:=ds.Employee.employer.relatedDataClass //$relClass1="Company"
$relClassN:=ds.Employee.directReports.relatedDataClass //$relClassN="Employee"
```

ORDA - DataStore

Para información sobre el objeto datastore, consulte la página ***Datastore*** *en la Guía del desarrollador 4D.*

ds.{dataclassName} New 17.0
ds New 17.0

ds.{dataclassName}

Parámetro	*Tipo*	*Descripción*
ds.{dataclassName}	*DataClass*	*Objeto DataClass*

Descripción

*Cada dataclass en el datastore está disponible como una propiedad del objeto **ds** . El objeto devuelto contiene una descripción de la dataclass.*

*Los objetos dataclass se benefician de los métodos específicos listados en el tema **ORDA - DataClass**.*

Ejemplo

```
C_OBJECT($emp;$sel)
$emp:=ds.Employee //$emp contiene la  dataclass Employee
$sel:=$emp.all() // obtiene una entity selection de todos los empleados

 //también puedes escribir directamente:
$sel:=ds.Employee.all()
```

ds

ds -> Resultado

Parámetro	Tipo		Descripción
Resultado	Objeto		Nueva referencia del almacén de datos

Descripción

*El comando **ds** devuelve una nueva referencia al almacén de datos que coincide con la base de datos 4D actual.*
***Nota**: el uso de **ds** requiere que su base cumpla con ORDA como se especifica en la sección **Prerequisitos ORDA**.*

*Este almacén de datos se abre automáticamente y está disponible directamente a través de **ds** . Se aplican las siguientes reglas:*

- *Un almacén de datos solo hace referencia a las tablas con una sola llave principal. Las tablas sin llave primaria o con llaves primarias compuestas no se referencian.*
- *Los atributos de tipo BLOB no se gestionan en el almacén de datos.*

*Para más información sobre la implementación del almacén de datos, consulte la sección **Datastore** .*

Ejemplo

Uso del almacén de datos actual de la base 4D :

```
$result:=ds.Employees.query("nom = :1";"S@")
```

ORDA - Entity

---

*Para saber cómo manejar las entidades en su código, consulte la página **Entidades** en la Guía del desarrollador 4D.*

- *entity.{attributeName}* New 17.0
- *entity.clone* New 17.0
- *entity.diff* New 17.0
- *entity.drop* New 17.0
- *entity.first* New 17.0
- *entity.fromObject* New 17.0
- *entity.getKey* New 17.0
- *entity.getSelection* New 17.0
- *entity.getStamp* New 17.0
- *entity.indexOf* New 17.0
- *entity.isNew* New 17.0
- *entity.last* New 17.0
- *entity.lock* New 17.0
- *entity.next* New 17.0
- *entity.previous* New 17.0
- *entity.reload* New 17.0
- *entity.save* New 17.0
- *entity.toObject* New 17.0
- *entity.touched* New 17.0
- *entity.touchedAttributes* New 17.0
- *entity.unlock* New 17.0

entity.{attributeName}

Parámetro	Tipo	Descripción
entity.{attributeName}	*Mixed*	*Valor actual del atributo en la entidad*

Descripción

Todo atributo dataclass está disponible como una propiedad de una entidad, que almacena el valor del atributo para la entidad.
Nota: *los atributos Dataclass también se pueden alcanzar usando la sintaxis alternativa utilizando [ ].*
El tipo de valor de atributo depende del tipo de atributo (relación o atributo):

- *Si el tipo nomAtributo es* ***storage****:*
  entity.attributeName *devuelve un valor del mismo tipo que nomAtributo.*
- *Si el tipo nomAtributo es* ***relatedEntity****:*
  entity.attributeName *devuelve la entidad relacionada. Los valores de la entidad relacionada están directamente disponibles a través de propiedades en cascada, por ejemplo "myEntity.employer.employees[0].lastname".*
- *Si el tipo nomAtributo es* ***relatedEntities****:*
  entity.attributeName *devuelve una nueva entity selection de entidades relacionadas. Los duplicados se eliminan (se devuelve una entity selection).*

Nota: *para más información sobre el tipo de atributo, consulte la descripción de la propiedad* ***dataClassAttribute.kind****.*

Ejemplo

```
C_OBJECT($myEntity)
$myEntity:=ds.Employee.new() //Crea un nuevo objeto de tipo entidad
$myEntity.name:="Dupont" // asignar 'Dupont' al atributo 'name'
$myEntity.firstname:="John" //asignar 'John' al atributo 'firstname'
$myEntity.save() //guardar la entidad
```

entity.clone( )

entity.clone () -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entity*		*Nueva entidad referenciando el registro*

Descripción

*El método **entity.clone( )** crea en la memoria una nueva entidad que hace referencia al mismo registro que la entidad original. Este método le permite actualizar entidades por separado. Tenga en cuenta que las modificaciones realizadas a las entidades se guardarán en el registro referenciado solo cuando se ejecute el método **entity.save( )**.*

Ejemplo

```
C_OBJECT(emp;empCloned)
emp:=ds.Employee.get(672)
empCloned:=emp.clone()

emp.lastName:="Smith" //Actualizaciones en $emp no se hacen en $empCloned
```

entity.diff( )

entity.diff (entidadAComparar {; atributosAComparar}) -> Resultado

Parámetro	Tipo		Descripción
entidadAComparar	Entity	➡	Entidad a comparar con la entidad original
atributosAComparar	Collection	➡	Nombre de los atributos a comparar
Resultado	Collection	⬅	Diferencias entre las entidades

Descripción

*El método **entity.diff( )** compara el contenido de dos entidades y devuelve sus diferencias.*

En entidadAComparar, pase la entidad que se comparará con la entidad original.

En atributosAComparar, puede designar atributos específicos a comparar. Si se da, la comparación se realiza solo en los atributos especificados. Si no se da, se devuelven todas las diferencias entre las entidades.

Las diferencias se devuelven como una colección de objetos cuyas propiedades son:

Nombre de la propiedad	Tipo	Descripción
attributeName	Cadena	Nombre del atributo
value	Depende del tipo de atributo	Valor del atributo en la entidad
otherValue	Depende del tipo de atributo	Valor del atributo en entidadAComparar

*Solo los atributos con diferentes valores están incluidos en la colección. Si no se encuentran diferencias, **entity.diff( )** devuelve una colección vacía.*

*El método aplica a las propiedades cuyo tipo es storage o relatedEntity (ver **dataClassAttribute.kind**). En el caso de que una entidad relacionada se haya actualizado (llave foránea), el nombre de la entidad relacionada y su nombre de llave primaria se devuelven como propiedades attributeName (value y otherValue están vacías para los nombres de entidad relacionados)*

Si una de las entidades comparadas es Null, se genera un error.

Ejemplo 1

```
C_COLLECTION($diff1;$diff2)
employee:=ds.Employee.query("ID=1001").first()
$clone:=employee.clone()
employee.firstName:="MARIE"
employee.lastName:="SOPHIE"
employee.salary:=500
$diff1:=$clone.diff(employee) // Todas las diferencias se devuelven
$diff2:=$clone.diff(employee;New collection"firstName";"lastName"))
 // Sólo se devuelven las diferencias en firstName y lastName
```

$diff1:

```
[ { "attributeName": "firstName", "value": "Natasha", "otherValue": "MARIE" }, { "attributeName": "lastName", "value": "Locke", "otherValue": "SOPHIE" }, { "attributeName": "salary", "value": 66600, "otherValue": 500 } ]
```

$diff2:

```
[ { "attributeName": "firstName", "value": "Natasha", "otherValue": "MARIE" }, { "attributeName": "lastName", "value": "Locke", "otherValue": "SOPHIE" } ]
```

Ejemplo 2

```
vCompareResult1:=New collection
vCompareResult2:=New collection
vCompareResult3:=New collection
$attributesToInspect:=New collection

$e1:=ds.Employee.get(636)
$e2:=ds.Employee.get(636)

$e1.firstName:=$e1.firstName+" update"
$e1.lastName:=$e1.lastName+" update"

$c:=ds.Company.get(117)
$e1.employer:=$c
$e2.salary:=100

$attributesToInspect.push("firstName")
```

```
$attributesToInspect.push("lastName")

vCompareResult1:=$e1.diff($e2)
vCompareResult2:=$e1.diff($e2;$attributesToInspect)
vCompareResult3:=$e1.diff($e2;$e1.touchedAttributes())
```

vCompareResult1 (todas las diferencias se devuelven):

[{ "attributeName": "firstName", "value": "Karla update", "otherValue": "Karla" }, { "attributeName": "lastName", "value": "Marrero update", "otherValue": "Marrero" }, { "attributeName": "salary", "value": 33500, "otherValue": 100 }, { "attributeName": "employerID", "value": 117, "otherValue": 118 }, { "attributeName": "employer", "value": "[object Entity]",// Entity 117 from Company "otherValue": "[object Entity]"// Entity 118 from Company }]

vCompareResult2 (solo se devuelven las diferencias en $attributesToInspect)

[{ "attributeName": "firstName", "value": "Karla update", "otherValue": "Karla" }, { "attributeName": "lastName", "value": "Marrero update", "otherValue": "Marrero" }]

vCompareResult3 (solo se devuelven las diferencias en $e1 touched attributes)

[{ "attributeName": "firstName", "value": "Karla update", "otherValue": "Karla" }, { "attributeName": "lastName", "value": "Marrero update", "otherValue": "Marrero" }, { "attributeName": "employerID", "value": 117, "otherValue": 118 }, { "attributeName": "employer", "value": "[object Entity]",// Entidad 117 de Company "otherValue": "[object Entity]"// Entidad 118 de Company }]

entity.drop( )

entity.drop ({modo}) -> Resultado

Parámetro	Tipo		Descripción
modo	Entero largo	➡	dk force drop if stamp changed: fuerza la caída incluso si el sello ha cambiado mientras tanto
Resultado	Objeto	⮌	Resultado de la operación de soltar

Descripción

*El método **entity.drop( )** elimina del datastore los datos contenidos en la entidad, en la tabla relacionada con su dataClass. Tenga en cuenta que la entidad permanece en la memoria.*

*En una aplicación multiusuario o multiproceso, el método **entity.drop( )** se ejecuta bajo un mecanismo "bloqueo optimista", mientras que un sello de bloqueo interno se incrementa automáticamente cada vez que se guarda el registro. Para más información, consulte la página **Bloquear entidades**.*

Por defecto, si se omite el parámetro modo, el método devolverá un error (ver abajo) si la misma entidad fue modificada (es decir, el sello ha cambiado) por otro proceso o usuario mientras tanto.
De lo contrario, puede pasar la opción dk force drop if stamp changed en el parámetro modo: en este caso, la entidad se suelta incluso si el sello ha cambiado (y la llave principal sigue siendo la misma).

Resultado

*El objeto devuelto por **entity.drop( )** contiene las siguientes propiedades:*

Propiedad		Tipo	Descripción
success		booleano	true si la acción de guardar es exitosa, de lo contrario es false.
			Disponible solo si se utiliza la opción dk auto merge:
autoMerged		booleano	true si se realizó una fusión automática, en caso contrario, false.
			Disponible solo en caso de error:
status(*)		número	Código de error, ver a continuación
statusText(*)		texto	Descripción del error, ver a continuación
			Disponible solo en caso de error de bloqueo pesimista:
LockKindText		texto	"Bloqueado por registro"
lockInfo		objeto	Información sobre el origen del bloqueo
	task_id	número	ID del proceso
	user_name	texto	Nombre de usuario de sesión en la máquina
	user4d_id	texto	Nombre de usuario en el directorio de la base de datos 4D
	host_name	texto	Nombre del equipo
	task_name	texto	Nombre del proceso
	client_version	texto	
			Disponible solo en caso de error grave *(error grave al tratar de duplicar una llave primaria, disco lleno...):*
errors		colección de objetos	
	message	texto	Mensaje de error
	component signature	texto	Firma del componente interno (por ejemplo, "dmbg" representa el componente de la base)
	errCode	número	Código de error

(*) Los siguientes valores pueden devolverse en las propiedades status y statusText del objeto Resultado en caso de error:

Constante	Valor	Comentario
dk status entity does not exist anymore	5	*La entidad ya no existe en los datos. Este error puede ocurrir en los siguientes casos:* • *la entidad ha sido suprimida (el stamp ha cambiado y el espacio de memoria ahora está liberado)* • *la entidad ha sido suprimida y reemplazada por otra con una llave primaria diferente (el stamp ha cambiado y una nueva entidad ahora ocupa el espacio de memoria). Cuando se usa **entity.drop( )**, este error puede devolverse cuando se usa la opción dk force drop if stamp changed. Al usar **entity.lock( )**, este error puede devolverse cuando se usa la opción dk reload if stamp changed* ***statusText asociado**: "La entidad ya no existe"*
dk status locked	3	*La entidad está bloqueada por un bloqueo pesimista.* ***statusText asociado**: "Ya bloqueado" ("Already locked")*
dk status serious error	4	*Un error grave es un error de bajo nivel de la base de datos (por ejemplo, llave duplicada), un error de hardware, etc.* ***statusText asociado**: "Otro error" ("Other error")*
dk status stamp has changed	2	*El valor de stamp interno de la entidad no coincide con el de la entidad almacenada en los datos (bloqueo optimista).* • *con **entity.save( )**: error solo si no se utiliza la opción dk auto merge* • *con **entity.drop( )**: error solo si no se usa la opción dk force drop if stamp changed* • *con **entity.lock( )**: error solo si no se usa la opción dk reload if stamp changed* ***statusText asociado**: "El stamp ha cambiado" ("Stamp has changed")*

Ejemplo 1

Ejemplo sin la opción dk force drop if stamp changed:

```
C_OBJECT($employees;$employee;$status)
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$status:=$employee.drop()
Case of
    :($status.success)
        ALERT("You have dropped "+$employee.firstName+" "+$employee.lastName) //La entidad soltada permanece en la memoria
    :($status.status=dk status stamp has changed)
        ALERT($status.statusText)
End case
```

Ejemplo 2

Ejemplo con la opción dk force drop if stamp changed:

```
C_OBJECT($employees;$employee;$status)
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$status:=$employee.drop(dk force drop if stamp changed)
Case of
    :($status.success)
        ALERT("You have dropped "+$employee.firstName+" "+$employee.lastName) //La entidad soltada permanece en memoria
    :($status.status=dk status entity does not exist anymore)
        ALERT($status.statusText)
End case
```

entity.first( )

entity.first () -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entity, Null		Primera entidad de la entity selection de la entidad

Descripción

*El método **entity.first( )**devuelve una referencia a la entidad en la primera posición de la entity selection a la que pertenece la entidad original.*

*Si la entidad no pertenece a ninguna entity selection existente (es decir, **entity.getSelection( )** devuelve Null), el método devuelve un valor Null.*

Ejemplo

```
C_OBJECT($employees;$employee;$firstEmployee)
$employees:=ds.Employee.query("lastName = :1";"H@") //Esta entity selection contiene 3 entidades
$employee:=$employees[2]
$firstEmployee:=$employee.first() //$firstEmployee es la primera entidad de la entity selection $employees
```

entity.fromObject( )

entity.fromObject (objeto)

Parámetro	Tipo		Descripción
objeto	Objeto	➡	Objeto desde el cual llenar la entidad

Descripción

*El método **entity.fromObject( )** llena la entidad con el contenido objeto.*

***Nota:** este método modifica la entidad original.*

El mapeo entre el objeto y la entidad se realiza en los nombres de los atributos:

- *Si una propiedad del objeto no existe en la dataclass, se ignora.*
- *Los tipos de datos deben ser equivalentes. Si hay una discrepancia de tipo entre el objeto y la dataclass, 4D intenta convertir los datos siempre que sea posible (ver **Tipos de datos**), de lo contrario, el atributo quedará intacto.*

*objeto puede contener una **related entity** bajo las siguientes condiciones:*

- *objeto objeto contiene la llave foránea en sí, o*
- *objeto contiene una propiedad de tipo objeto con el mismo nombre que la entidad relativa, que contiene una sola propiedad llamada "__KEY".*
- *si la entidad relativa no existe, se ignora.*

Ejemplo

Con el objeto $o siguiente:

```
{ "firstName": "Mary", "lastName": "Smith", "salary": 36500, "birthDate": "1958-10-27T00:00:00.000Z", "woman": true, "managerID": 411,// relatedEntity ofrecida con la llave primaria "employerID": 20 // relatedEntity ofrecida con la llave primaria }
```

*El siguiente código creará una entidad con las entidades relativas **manager** y **employer**.*

```
C_OBJET($o)
$entity:=ds.Emp.new()
$entity.fromObject($o)
$entity.save()
```

Puede igualmente utilizar una entidad relativa ofrecida en forma de objeto:

```
{ "firstName": "Marie", "lastName": "Lechat", "salary": 68400, "birthDate": "1971-09-03T00:00:00.000Z", "woman": false, "employer": {// relatedEntity ofrecida en forma de objeto "__KEY": "21" }, "manager": {// relatedEntity fofrecida en forma de objeto "__KEY": "411" } }
```

entity.getKey( )

entity.getKey ({modo}) -> Resultado

Parámetro	*Tipo*		*Descripción*
modo	*Entero largo*	➡	*dk key as string: la llave primaria se devuelve como una cadena, cualquiera que sea el tipo de llave primaria*
Resultado	*Entero largo, Texto*	⮌	*Valor de la llave primaria de la entidad*

Descripción

*El método **entity.getKey( )** devuelve el valor de la llave primaria de la entidad.*
Las llaves primarias pueden ser números (entero largo) o textos. Puede "forzar" el método a devolver el valor de la llave primaria en forma de cadena, sin importar su tipo de origen, pasando la opción dk key as string en el parámetro modo.

Ejemplo

```
C_OBJECT($employees;$employee)
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees[0]
ALERT("The primary key is "+$employee.getKey(dk key as string))
```

entity.getSelection( )

entity.getSelection () -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*EntitySelection, Null*		*Entity selection a la cual pertenece la entidad*

Descripción

*El método **entity.getSelection( )** devuelve la entity selection a la que pertenece la entidad.*
Si la entidad no pertenece a una entity selection, el método devuelve Null.

Ejemplo

```
C_OBJECT($emp;$employees;$employees2)
$emp:=ds.Employee.get(672) // Esta entidad no pertenece a ninguna selección de entidad
$employees:=$emp.getSelection() // $employees es Null

$employees2:=ds.Employee.query("lastName=:1";"Smith") //Esta entity selection contiene 6 entidades
$emp:=$employees2[0] // Esta entidad pertenece a una entity selection

ALERT("The entity selection contains "+String($emp.getSelection().length)+" entities")
```

entity.getStamp( )

entity.getStamp () -> Resultado

Parámetro	Tipo		Descripción
Resultado	Número	➲	Stamp de la entidad (0 si la entidad acaba de ser creada)

Descripción

*El método **entity.getStamp( )** devuelve el valor actual del stamp de la entidad.*

*El stamp interno se incrementa automáticamente por 4D cada vez que se guarda la entidad. Gestiona el acceso concurrente de usuarios y modificaciones a las mismas entidades. Para más información sobre este mecanismo, consulte la página **Bloquear entidades**.*

***Nota:** para una nueva entidad (nunca guardada), el método devuelve 0. Sin embargo, para saber si una entidad acaba de crearse, se recomienda utilizar **entity.isNew( )**.*

Ejemplo

```
C_OBJET($entity)
C_LONGINT($stamp)

$entity:=ds.Employee.new()
$entity.lastname:="Smith"
$entity.save()
$stamp:=$entity.getStamp() //$stamp=1

$entity.lastname:="Wesson"
$entity.save()
$stamp:=$entity.getStamp() //$stamp=2
```

entity.indexOf( )

entity.indexOf ({entitySelection}) -> Resultado

Parámetro	Tipo		Descripción
entitySelection	EntitySelection	➡	La posición de la entidad se da de acuerdo con esta entity selection
Resultado	Entero largo	⮌	Posición de la entity en una entity selection

Descripción

*El método **entity.indexOf( )** devuelve la posición de la entidad en una selección de entidad.*

Por defecto, si se omite el parámetro entitySelection, el método devuelve la posición de la entidad dentro de su propia selección de entidades. De lo contrario, devuelve la posición de la entidad dentro de la entitySelection especificada.

El valor resultante se incluye entre 0 y la longitud de la entity selection -1.

- *Si la entidad no tiene una entity selection o no pertenece a entitySelection, el método devuelve -1.*
- *Si entitySelection es Null o no pertenece a la misma dataclass que la entidad, se genera un error.*

Ejemplo

```
C_OBJECT($employees;$employee)
$employees:=ds.Employee.query("lastName = :1";"H@") //Esta entity selection contiene 3 entidades
$employee:=$employees[1] //Esta entidad pertenece a una selección de entidad
ALERT("The index of the entity in its own entity selection is "+String($employee.indexOf())) //1

C_OBJECT($employee)
$employee:=ds.Employee.get(725) //Esta entidad no pertenece a una entity selection
ALERT("The index of the entity is "+String($employee.indexOf())) // -1
```

entity.isNew( )

entity.isNew () -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*	⮌	*True si la entidad acaba de crearse y aún no está guardada. De lo contrario, False.*

Descripción

*El método **entity.isNew( )** devuelve true si la entidad a la que se aplica acaba de crearse y aún no se ha guardado en el datastore. De lo contrario, devuelve False.*

Ejemplo

```
C_OBJECT($emp)

$emp:=ds.Employee.new()

If($emp.isNew())
    ALERT("This is a new entity")
End if
```

entity.last( )

entity.last () -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entity, Null*		*Última entidad de la entity selection de la entidad*

Descripción

*El método **entity.last( )** devuelve una referencia a la entidad en la última posición de la entity selection a la que pertenece la entidad original.*
*Si la entidad no pertenece a ninguna entity selection existente (es decir **entity.getSelection( )** devuelve Null), el método devuelve un valor Null.*

Ejemplo

```
C_OBJECT($employees;$employee;$lastEmployee)
$employees:=ds.Employee.query("lastName = :1";"H@") //Esta entity selection contiene 3 entidades
$employee:=$employees[0]
$lastEmployee:=$employee.last() //$lastEmployee es la última entidad de la entity selection $employees
```

entity.lock( )

entity.lock ({modo}) -> Resultado

Parámetro	Tipo		Descripción
modo	Entero largo	➡	dk reload if stamp changed: recarga antes de bloquear si stamp cambió
Resultado	Objeto	⮌	Resultado de la operación de bloqueo

Descripción

*El método **entity.lock( )**pone un bloqueo pesimista(*) en el registro al que hace referencia la entidad. El bloqueo se establece para un registro y para todas las referencias de la entidad en el proceso actual.*

*(*)Para más información, consulte la página **Bloquear entidades**.*

*Otros procesos verán este registro como bloqueado (la propiedad **result.success** contendrá false si tratan de bloquear la misma entidad usando este método. Solo los métodos ejecutados en la sesión de "bloqueo" pueden editar y guardar los atributos de la entidad. La entidad puede ser cargada por otras sesiones, pero no podrán ingresar ni guardar valores*

El registro bloqueado está desbloqueado:

- *cuando se llama al método **unlock()** en una entidad coincidente en el mismo proceso*
- *automáticamente, cuando ya no se hace referencia a ninguna entidad en la memoria. Por ejemplo, si el bloqueo es solo una referencia local de una entidad, la entidad se desbloquea cuando el método finaliza. Siempre que haya referencias a la entidad en la memoria, el registro queda bloqueado.*

De forma predeterminada, si se omite el parámetro modo, el método devolverá un error (ver a continuación) si la misma entidad fue modificada (es decir, el sello ha cambiado) por otro proceso o usuario mientras tanto.
De lo contrario, puede pasar la opción dk reload if stamp changed en el parámetro modo: en este caso, no se devuelve ningún error y la entidad se recarga cuando el stamp ha cambiado (si la entidad todavía existe y la llave primaria sigue siendo la misma).

Resultado

*El objeto devuelto por **entity.lock( )** contiene las siguientes propiedades:*

Propiedad		Tipo	Descripción
success		booleano	true si la acción de guardar es exitosa (o si la entidad ya está bloqueada en el proceso actual), de lo contrario false.
			Disponible solo si se utiliza la opción dk reload if stamp changed:
wasReloaded		booleano	true si la entidad fue recargada con éxito, de lo contrario false.
			Disponible solo en caso de error:
status(*)		número	Código de error, ver a continuación
statusText(*)		texto	Descripción del error, ver a continuación
			Disponible solo en caso de error de bloqueo pesimista:
LockKindText		texto	"Bloqueado por registro"
lockInfo		objeto	Información sobre el origen del bloqueo
	task_id	número	ID del proceso
	user_name	texto	Nombre de usuario de sesión en la máquina
	user4d_id	texto	Nombre de usuario en el directorio de la base de datos 4D
	host_name	texto	Nombre del equipo
	task_name	texto	Nombre del proceso
	client_version	texto	
			Disponible solo en caso de error grave (llave primaria ya existe, disco lleno...):
errors		colección de objetos	
	message	texto	Mensaje de error
	component signature	texto	Firma del componente interno (por ejemplo, "dmbg" representa el componente de la base)
	errCode	number	Código de error

(*) Los siguientes valores pueden devolverse en las propiedades status y statusText del objeto Resultado en caso de error:

Constante	Valor	Comentario
dk status entity does not exist anymore	*5*	*La entidad ya no existe en los datos. Este error puede ocurrir en los siguientes casos:* • *la entidad ha sido suprimida (el stamp ha cambiado y el espacio de memoria ahora está liberado)* • *la entidad ha sido suprimida y reemplazada por otra con una llave primaria diferente (el stamp ha cambiado y una nueva entidad ahora ocupa el espacio de memoria). Cuando se usa **entity.drop( )**, este error puede devolverse cuando se usa la opción dk force drop if stamp changed. Al usar **entity.lock( )**, este error puede devolverse cuando se usa la opción dk reload if stamp changed* ***statusText asociado**: "La entidad ya no existe"*
dk status locked	*3*	*La entidad está bloqueada por un bloqueo pesimista.* ***statusText asociado**: "Ya bloqueado" ("Already locked")*
dk status serious error	*4*	*Un error grave es un error de bajo nivel de la base de datos (por ejemplo, llave duplicada), un error de hardware, etc.* ***statusText asociado**: "Otro error" ("Other error")*
dk status stamp has changed	*2*	*El valor de stamp interno de la entidad no coincide con el de la entidad almacenada en los datos (bloqueo optimista).* • *con **entity.save( )**: error solo si no se utiliza la opción dk auto merge* • *con **entity.drop( )**: error solo si no se usa la opción dk force drop if stamp changed* • *con **entity.lock( )**: error solo si no se usa la opción dk reload if stamp changed* ***statusText asociado**: "El stamp ha cambiado" ("Stamp has changed")*

Ejemplo 1

Ejemplo con error:

```
C_OBJECT($employee;$status)
$employee:=ds.Employee.get(716)
$status:=$employee.lock()
Case of
   :($status.success)
      ALERT("You have locked "+$employee.firstName+" "+$employee.lastName)
   :($status.status=dk status stamp has changed)
      ALERT($status.statusText)
End case
```

Ejemplo 2

Ejemplo con la opción dk reload if stamp changed:

```
C_OBJECT($employee;$status)

$employee:=ds.Employee.get(717)
$status:=$employee.lock(dk reload if stamp changed)
Case of
   :($status.success)
      ALERT("You have locked "+$employee.firstName+" "+$employee.lastName)
   :($status.status=dk status entity does not exist anymore)
      ALERT($status.statusText)
End case
```

entity.next( )

entity.next () -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entity, Null*		*Referencia a la siguiente entidad en la entity selection*

Descripción

*El método **entity.next( )** devuelve una referencia a la siguiente entidad en la entity selection a la que pertenece la entidad original.*

*Si la entidad no pertenece a ninguna entity selection existente (es decir, **entity.getSelection( )** devuelve Null), el método devuelve un valor Null.*

Si no hay una próxima entidad válida en la entity selection (es decir, usted está en la última entidad de la selección), el método devuelve Null. Si se ha eliminado la siguiente entidad, el método devuelve la siguiente entidad válida (y eventualmente Null).

Ejemplo

```
C_OBJECT($employees;$employee;$nextEmployee)
$employees:=ds.Employee.query("lastName = :1";"H@") //Esta entity selection contiene 3 entidades
$employee:=$employees[0]
$nextEmployee:=$employee.next() //$nextEmployee es la segunda entidad de la entity selection $employees
```

entity.previous( )

entity.previous () -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entity, Null*	➲	*Devuelve la entidad anterior de una entity selection*

Descripción

*El método **entity.previous( )** devuelve una referencia a la entidad anterior en la entity selection a la que pertenece la entidad original.*

*Si la entidad no pertenece a ninguna entity selection existente (es decir, **entity.getSelection( )** devuelve Null), el método devuelve un valor Null.*

Si no hay una entidad válida anterior en la entity selection (es decir, está en la primera entidad de la selección), el método devuelve Null. Si la entidad anterior se ha eliminado, el método devuelve la entidad válida anterior (y eventualmente Null).

Ejemplo

```
C_OBJECT($employees;$employee;$previousEmployee)
$employees:=ds.Employee.query("lastName = :1";"H@") //Esta entity selection contiene 3 entidades
$employee:=$employees[1]
$previousEmployee:=$employee.previous() //$previousEmployee es la primera entidad de la entity selection $employees
```

entity.reload( )

entity.reload () -> Resultado

Parámetro	Tipo	Descripción
Resultado	Objeto	Estado objeto

Descripción

*El método **entity.reload( )** recarga el contenido de la entidad en memoria, de acuerdo a la información almacenada en la tabla relacionada con el dataclass en el datastore. La recarga se realiza sólo si la entidad existe con la misma llave primaria.*

Resultado

*El objeto devuelto por **entity.reload( )** contiene las siguientes propiedades:*

Propiedad	Tipo	Descripción
success	booleano	true si la acción de guardar es exitosa, de lo contrario false.
		Disponible solo en caso de error:
status(*)	número	Código de error, ver abajo
statusText(*)	texto	Descripción del error, ver abajo

(*) Los siguientes valores pueden devolverse en las propiedades status y statusText del objeto Resultado en caso de error:

Constante	Valor	Comentario
dk status entity does not exist anymore	5	La entidad ya no existe en los datos. Este error puede ocurrir en los siguientes casos: • la entidad ha sido suprimida (el stamp ha cambiado y el espacio de memoria ahora está liberado) • la entidad ha sido suprimida y reemplazada por otra con una llave primaria diferente (el stamp ha cambiado y una nueva entidad ahora ocupa el espacio de memoria). Cuando se usa **entity.drop( )**, este error puede devolverse cuando se usa la opción dk force drop if stamp changed. Al usar **entity.lock( )**, este error puede devolverse cuando se usa la opción dk reload if stamp changed **statusText asociado**: "La entidad ya no existe"
dk status serious error	4	Un error grave es un error de bajo nivel de la base de datos (por ejemplo, llave duplicada), un error de hardware, etc. **statusText asociado**: "Otro error" ("Other error")

Ejemplo

```
C_OBJECT($employee;$employees;$result)

$employees:=ds.Employee.query("lastName=:1";"Hollis")
$employee:=$employees[0]
$employee.firstName:="Mary"
$result:=$employee.reload()
Case of
   :($result.success)
      ALERT("Reload has been done")
   :($result.status=dk status entity does not exist anymore)
      ALERT("The entity has been dropped")
End case
```

entity.save( )

entity.save ({modo}) -> Resultado

Parámetro	Tipo		Descripción
modo	Entero largo	➡	dk auto merge: habilita el modo de fusión automática
Resultado	Objeto	⮌	Resultado de guardar la operación

Descripción

*El método **entity.save( )** guarda los cambios realizados en la entidad en la tabla relacionada con su dataClass. Debe llamar a este método después de crear o modificar una entidad si desea guardar los cambios realizados en ella.*

*La operación de guardar se ejecuta solo si se ha "tocado" al menos un atributo de entidad (ver los métodos **entity.touched( )** y **entity.touchedAttributes( )**). De lo contrario, el método no hace nada (no se llama al activador).*

*En una aplicación multiusuario o multiproceso, el método **entity.save( )** se ejecuta bajo un mecanismo de "bloqueo optimista", mientras que un sello de bloqueo interno se incrementa automáticamente cada vez que se guarda el registro. Para más información, consulte la página **Bloquear entidades**.*

Por defecto, si se omite el parámetro modo, el método devolverá un error (ver abajo) siempre que la misma entidad haya sido modificada por otro proceso o usuario mientras tanto, sean cuales sean los atributos modificados.
De lo contrario, puede pasar la opción dk auto merge en el parámetro modo: cuando el modo de fusión automática está habilitado, una modificación realizada por otro proceso/usuario en la misma entidad pero en un atributo diferente no dará lugar a un error. Los datos resultantes guardados en la entidad serán la combinación (la "fusión") de todas las modificaciones no concurrentes (si se aplicaron modificaciones al mismo atributo, la operación de guardado falla y se devuelve un error, incluso con el modo de fusión automática).

***Nota:** el modo de fusión automática no está disponible para los atributos de tipo imagen, objeto y texto cuando se almacenan fuera del registro. Los cambios concurrentes en estos atributos darán lugar a un error dk status stamp has changed.*

Resultado

*El objeto devuelto por **entity.save( )** contiene las siguientes propiedades:*

Propiedad		Tipo	Descripción
success		booleano	true si la acción de guardar es exitosa, de lo contrario es false.
			Disponible solo si se utiliza la opción dk auto merge:
autoMerged		booleano	true si se realizó una fusión automática, en caso contrario, false.
			Disponible solo en caso de error:
status(*)		número	Código de error, ver a continuación
statusText(*)		texto	Descripción del error, ver a continuación
			Disponible solo en caso de error de bloqueo pesimista:
LockKindText		texto	"Bloqueado por registro"
lockInfo		objeto	Información sobre el origen del bloqueo
	task_id	número	ID del proceso
	user_name	texto	Nombre de usuario de sesión en la máquina
	user4d_id	texto	Nombre de usuario en el directorio de la base de datos 4D
	host_name	texto	Nombre del equipo
	task_name	texto	Nombre del proceso
	client_version	texto	
			Disponible solo en caso de error grave *(error grave al tratar de duplicar una llave primaria, disco lleno ...):*
errors		colección de objetos	
	message	texto	Mensaje de error
	component signature	texto	firma del componente interno (por ejemplo, "dmbg" representa el componente de la base)
	errCode	número	Código de error

(*) Los siguientes valores pueden devolverse en las propiedades status y statusText el objeto Resultado en caso de error:

Constante	Valor	Comentario
dk status automerge failed	6	(Solo si se usa la opción dk auto merge) La opción de combinación automática falló al guardar la entidad. **statusText asociado**: "Auto merge failed"
dk status entity does not exist anymore	5	La entidad ya no existe en los datos. Este error puede ocurrir en los siguientes casos: • la entidad ha sido suprimida (el stamp ha cambiado y el espacio de memoria ahora está liberado) • la entidad ha sido suprimida y reemplazada por otra con una llave primaria diferente (el stamp ha cambiado y una nueva entidad ahora ocupa el espacio de memoria). Cuando se usa **entity.drop( )**, este error puede devolverse cuando se usa la opción dk force drop if stamp changed. Al usar **entity.lock( )**, este error puede devolverse cuando se usa la opción dk reload if stamp changed **statusText asociado**: "La entidad ya no existe"
dk status locked	3	La entidad está bloqueada por un bloqueo pesimista. **statusText asociado**: "Ya bloqueado" ("Already locked")
dk status serious error	4	Un error grave es un error de bajo nivel de la base de datos (por ejemplo, llave duplicada), un error de hardware, etc. **statusText asociado**: "Otro error" ("Other error")
dk status stamp has changed	2	El valor de stamp interno de la entidad no coincide con el de la entidad almacenada en los datos (bloqueo optimista). • con **entity.save( )**: error solo si no se utiliza la opción dk auto merge • con **entity.drop( )**: error solo si no se usa la opción dk force drop if stamp changed • con **entity.lock( )**: error solo si no se usa la opción dk reload if stamp changed **statusText asociado**: "El stamp ha cambiado" ("Stamp has changed")

Ejemplo 1

Crear una nueva entity:

```
C_OBJECT($status;$employee)
$employee:=ds.Employee.new()
$employee.firstName:="Mary"
$employee.lastName:="Smith"
$status:=$employee.save()
If($status.success)
    ALERT("Employee created")
End if
```

Ejemplo 2

Actualizar una entidad sin la opción dk auto merge:

```
C_OBJECT($status;$employees;$employee)
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$employee.lastName:="Mac Arthur"
$status:=$employee.save()
Case of
    :($status.success)
        ALERT("Employee updated")
    :($status.status=dk status stamp has changed)
        ALERT($status.statusText)
End case
```

Ejemplo 3

Actualizar una entidad con la opción dk auto merge:

```
C_OBJECT($status;$employees;$employee)

$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees.first()
$employee.lastName:="Mac Arthur"
$status:=$employee.save(dk auto merge)
Case of
    :($status.success)
        ALERT("Employee updated")
    :($status.status=dk status automerge failed)
```

```
        ALERT($status.statusText)
    End case
```

entity.toObject( )

entity.toObject (filtro {; opciones}) -> Resultado

Parámetro	Tipo		Descripción
filtro	Cadena, Collection	➡	Atributo(s) a extraer
opciones	Entero largo	➡	dk with primary key: añade la propiedad _KEY; dk with stamp: añade la propiedad _STAMP
Resultado	Objeto	⮌	Objeto generado desde la entidad

Descripción

*El método **entity.toObject( )** devuelve un objeto que se ha creado a partir de la entidad. Los nombres de las propiedades en el objeto correspondiente a los nombres de los atributos de la entidad.*

En el parámetro filtro, pase los atributos de entidad a extraer. Se permiten dos sintaxis:

- *una cadena con rutas de propiedades separadas por comas: "propertyPath1, propertyPath2, ...".*
- *una colección de cadenas: ["propertyPath1","propertyPath2";...]*

*Si filtro se especifica para atributos del tipo **relatedEntity**:*

- *propertyPath = "relatedEntity" -> se extrae de forma simple: un objeto con propiedad __KEY (llave primaria).*
- *propertyPath = "relatedEntity.*" -> se extraen todas las propiedades*
- *propertyPath = "relatedEntity.propertyName1; relatedEntity.propertyName2; ..." -> solo se extraen esas propiedades*

*Si filtro se especifica para atributos del tipo **relatedEntities**:*

- *propertyPath = "relatedEntities.*" -> se extraen todas las propiedades*
- *propertyPath = "relatedEntities.propertyName1; relatedEntities.propertyName2; ..." -> solo se extraen esas propiedades*

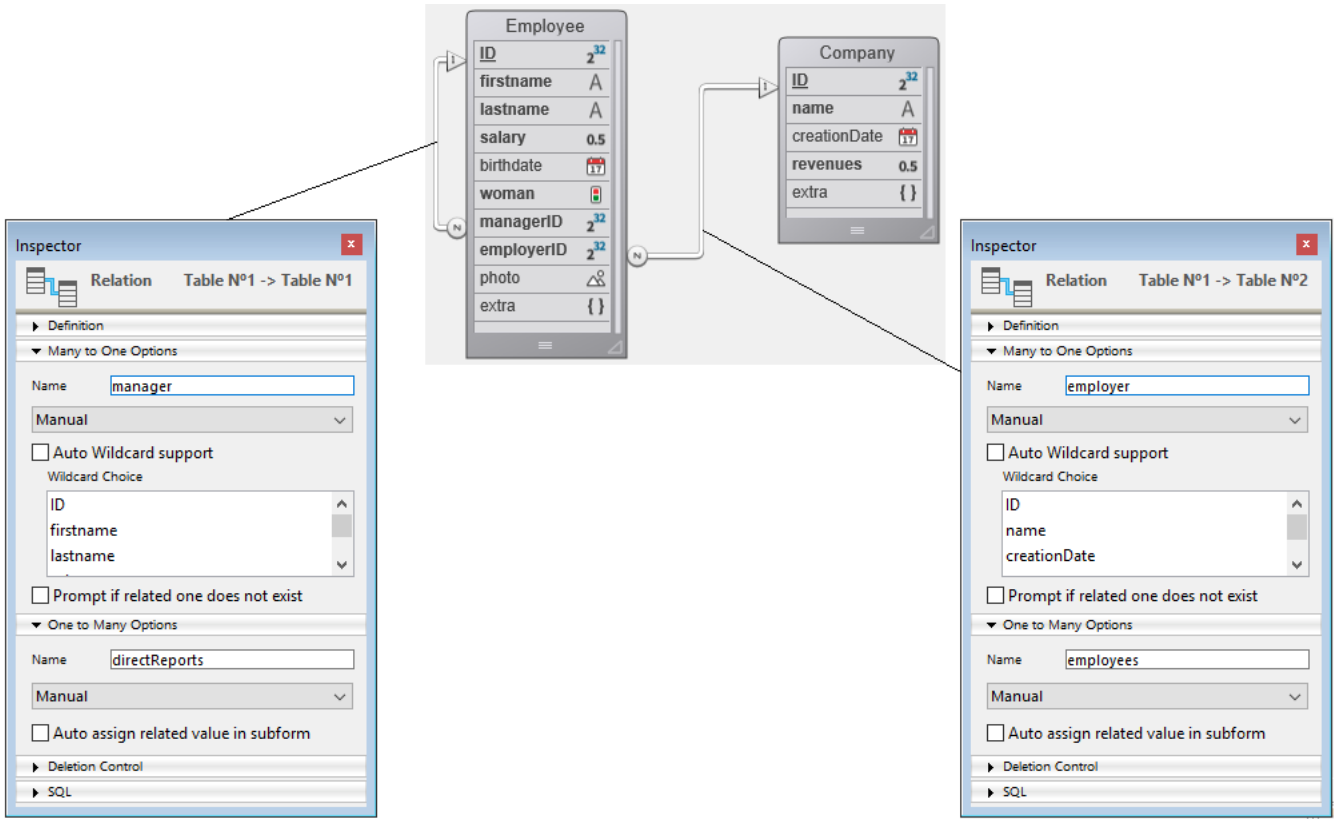
Si el parámetro filtro contiene una cadena vacía o "", el objeto devuelto contendrá:*

- *todos los atributos de tipo storage de la entidad*
- *atributos del tipo **relatedEntity**: se obtiene una propiedad con el mismo nombre que la entidad relacionada (nombre del enlace muchos a uno). El atributo se extrae de forma simple.*
- *atributos del tipo **relatedEntities**: atributo no revuelto.*

En el parámetro opciones, puede pasar los selectore dk with primary key y/o dk with stamp para agregar las llaves primarias y/o los stamps en los objetos extraídos.

Ejemplo 1

La estructura siguiente será utilizada para los ejemplos de esta sección:



Sin parámetro filtro:

```
employeeObject:=employeeSelected.toObject()
```

Devuelve:

```
{ "ID": 413, "firstName": "Greg", "lastName": "Wahl", "salary": 0, "birthDate": "1963-02-01T00:00:00.000Z", "woman": false, "managerID": 412, "employerID": 20, "photo": "[object Picture]", "extra": null, "employer": { // relatedEntity extraído de forma simple "__KEY": 20 }, "manager": { "__KEY": 412 } }
```

Ejemplo 2

Extracción de la llave primaria y del stamp:

```
employeeObject:=employeeSelected.toObject("";dk with primary key+dk with stamp)
```

Devuelve:

```
{ "__KEY": 413, "__STAMP": 1, "ID": 413, "firstName": "Greg", "lastName": "Wahl", "salary": 0, "birthDate": "1963-02-01T00:00:00.000Z", "woman": false, "managerID": 412, "employerID": 20, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 20 }, "manager": { "__KEY": 412 } }
```

Ejemplo 3

Extracción completa de los atributos de las relatedEntities:

```
employeeObject:=employeeSelected.toObject("directReports.*")
```

```
{ "directReports": [ { "ID": 418, "firstName": "Lorena", "lastName": "Boothe", "salary": 44800, "birthDate": "1970-10-02T00:00:00.000Z", "woman": true, "managerID": 413, "employerID": 20, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 20 }, "manager": { "__KEY": 413 } }, { "ID": 419, "firstName": "Drew", "lastName": "Caudill", "salary": 41000, "birthDate": "2030-01-12T00:00:00.000Z", "woman": false, "managerID": 413, "employerID": 20, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 20 }, "manager": { "__KEY": 413 } }, { "ID": 420, "firstName": "Nathan", "lastName": "Gomes", "salary": 46300, "birthDate": "2010-05-29T00:00:00.000Z", "woman": false, "managerID": 413, "employerID": 20, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 20 }, "manager": { "__KEY": 413 } } ] }
```

Ejemplo 4

Extracción de algunos atributos de las relatedEntities:

```
employeeObject:=employeeSelected.toObject("firstName, directReports.lastName")
```

Devuelve:

```
{ "firstName": "Greg", "directReports": [ { "lastName": "Boothe" }, { "lastName": "Caudill" }, { "lastName": "Gomes" } ] }
```

Ejemplo 5

Extracción de una relatedEntity de forma simple:

```
$coll:=New collection("firstName";"employer")
employeeObject:=employeeSelected.toObject($coll)
```

Devuelve:

```
{ "firstName": "Greg", "employer": { "__KEY": 20 } }
```

Ejemplo 6

Extracción de todos los atributos de una relatedEntity:

```
employeeObject:=employeeSelected.toObject("employer.*")
```

Devuelve:

```
{ "employer": { "ID": 20, "name": "India Astral Secretary", "creationDate": "1984-08-25T00:00:00.000Z", "revenues": 12000000, "extra": null } }
```

Ejemplo 7

Extracción de algunos atributos de una relatedEntity:

```
$col:=Creer collection
$col.push("employer.name")
$col.push("employer.revenues")
employeeObject:=employeeSelected.toObject($col)
```

Devuelve:

```
{ "employer": { "name": "India Astral Secretary", "revenues": 12000000 } }
```

entity.touched( )

entity.touched () -> Resultado

Parámetro	Tipo	Descripción
Resultado	Booleano	True si al menos un atributo de entidad ha sido modificado y aún no se ha guardado, de lo contrario es false

Descripción

*El método **entity.touched( )** indica si un atributo de la entidad se ha modificado o no desde que la entidad se cargó en memoria o guardó.*

Si un atributo ha sido modificado o calculado, el método devuelve true, de lo contrario devuelve false. Puede usar este método para averiguar si necesita guardar la entidad.

*Este método devuelve false para una entidad nueva (creada con **dataClass.new( )**). Sin embargo, tenga en cuenta que si utiliza un método que calcula un atributo de la entidad, el método **entity.touched( )** devolverá true. Por ejemplo, si llama a **entity.getKey( )** para calcular la llave primaria, el método **entity.touched( )** devuelve True.*

Ejemplo

En este ejemplo, verificamos si es necesario guardar la entidad:

```
C_OBJECT($emp)
$emp:=ds.Employee.get(672)
$emp.firstName:=$emp.firstName // Incluso si se actualiza con el mismo valor, el atributo se marca como tocado

If($emp.touched()) //si al menos uno de los atributos ha sido cambiado
    $emp.save()
End if // de lo contrario, no es necesario guardar la entidad
```

entity.touchedAttributes( )

entity.touchedAttributes -> Resultado

Parámetro	Tipo		Descripción
Resultado	Collection	➲	Nombres de los atributos tocados, o colección vacía

Descripción

*El método **entity.touchedAttributes( )** devuelve los nombres de los atributos que se han modificado desde que la entidad se cargó en la memoria.*

*Esto aplica a los atributos de tipo storage o relatedEntity (ver **dataClassAttribute.kind**).*

En el caso de que se haya tocado una entidad relacionada (es decir, llave foránea), se devuelve el nombre de la entidad relacionada y su llave primaria.
Si no se ha tocado ningún atributo de entidad, el método devuelve una colección vacía.

Ejemplo 1

```
C_COLLECTION($touchedAttributes)
C_OBJECT($emp)

$touchedAttributes:=New collection
$emp:=ds.Employee.get(725)
$emp.firstName:=$emp.firstName  //Incluso si se actualiza con el mismo valor, el atributo se marca como tocado
$emp.lastName:="Martin"
$touchedAttributes:=$emp.touchedAttributes()
 //$touchedAttributes: ["firstName","lastName"]
```

Ejemplo 2

```
C_COLLECTION($touchedAttributes)
C_OBJECT($emp;$company)

$touchedAttributes:=New collection

$emp:=ds.Employee.get(672)
$emp.firstName:=$emp.firstName
$emp.lastName:="Martin"

$company:=ds.Company.get(121)
$emp.employer:=$company

$touchedAttributes:=$emp.touchedAttributes()

 //collection $touchedAttributes: ["firstName","lastName","employer","employerID"]
```

En este caso:

- *firstName y lastName tienen un tipo storage*
- *employer tiene un tipo relatedEntity*
- *employerID es la llave externa de la entidad relacionada con el empleador*

entity.unlock( )

entity.unlock () -> Resultado

Parámetro	Tipo		Descripción
Resultado	Objeto		Estado objeto

Descripción

*El método **entity.unlock( )** elimina el bloqueo pesimista(*) en el registro que coincide con la entidad en el almacén de datos y la tabla relacionada con su dataclass.*

*(*)Para más información, consulte la página **Bloquear entidades**.*

Un registro se desbloquea automáticamente cuando ya no está referenciado por ninguna entidad en el proceso de bloqueo (por ejemplo: si el bloqueo se coloca en una referencia local de una entidad, la entidad y, por lo tanto, el registro se desbloquea cuando finaliza el proceso).

***Nota**: cuando un registro está bloqueado, debe ser desbloqueado desde el proceso de bloqueo y en la referencia de la entidad que pone el bloqueo. Por ejemplo:*

```
$e1:=dataClass.all()[0]
$e2:=dataClass.all()[0]
$e1.lock() // el desbloqueo debe hacerse en $e1. El desbloqueo en $e2 fallará.
```

Resultado

*El objeto devuelto por **entity.unlock( )** contiene las siguientes propiedades:*

Propiedad	Tipo	Descripción
success	booleano	true si la acción de guardar es exitosa, de lo contrario false. Si el desbloqueo se hace en la entidad soltada, en un registro no bloqueado, o en un registro bloqueado por otro proceso u otra entidad, success es false.

Ejemplo

```
C_OBJECT($employee;$status)

$employee:=ds.Employee.get(725)
$status:=$employee.lock()
... //processing
$status:=$employee.unlock()
If($status.success)
    ALERT("The entity is now unlocked")
End if
```

ORDA - EntitySelection

- *entitySelection.{attributeName}* New 17.0
- *entitySelection.length* New 17.0
- *entitySelection.queryPath* New 17.0
- *entitySelection.queryPlan* New 17.0
- *entitySelection[index]* New 17.0
- *Create entity selection* New 17.0
- *entitySelection.add* New 17.0
- *entitySelection.and* New 17.0
- *entitySelection.average* New 17.0
- *entitySelection.contains* New 17.0
- *entitySelection.count* New 17.0
- *entitySelection.distinct* New 17.0
- *entitySelection.drop* New 17.0
- *entitySelection.first* New 17.0
- *entitySelection.isOrdered* New 17.0
- *entitySelection.last* New 17.0
- *entitySelection.max* New 17.0
- *entitySelection.min* New 17.0
- *entitySelection.minus* New 17.0
- *entitySelection.or* New 17.0
- *entitySelection.orderBy* New 17.0
- *entitySelection.query* New 17.0
- *entitySelection.slice* New 17.0
- *entitySelection.sum* New 17.0
- *entitySelection.toCollection* New 17.0
- *USE ENTITY SELECTION* New 17.0

entitySelection.{attributeName}

Parámetro	Tipo	Descripción
entitySelection.{attributeName}	*Collection, EntitySelection*	*Proyección de valores de atributo para la selección de entidad*

Descripción

Todo atributo de clase de datos se puede usar como una propiedad de una selección de entidad para devolver una "proyección" de valores para el atributo en la selección de entidad. Los valores proyectados pueden ser una colección o una nueva selección de entidad, según el tipo (almacenamiento o relación) del atributo.

- *Si la clase de nomAtributo es **storage**:*
  ***entitySelection.attributeName** devuelve una colección de valores del mismo tipo que nomAtributo.*
- *Si la clase de nomAtributo es **relatedEntity**:*
  ***entitySelection.attributeName** devuelve una nueva selección de entidad de valores relacionados del mismo tipo que nomAtributo. Los duplicados se eliminan (se devuelve una selección de entidad desordenada).*
- *Si la clase de nomAtributo es **relatedEntities**:*
  ***entitySelection.attributeName** devuelve una nueva selección de entidad de valores relacionados del mismo tipo que nomAtributo. Los duplicados se eliminan (se devuelve una selección de entidad desordenada).*

Cuando un atributo de relación se usa como una propiedad de una selección de entidad, el resultado es siempre otra selección de entidad, incluso si solo se devuelve una entidad. En este caso, si no se devuelven entidades, el resultado es una selección de entidad vacía.

***Nota:** para más información sobre el tipo de atributo, consulte la descripción de la propiedad **dataClassAttribute.kind**.*

Ejemplo 1

Proyección de valores de almacenamiento:

```
C_COLLECTION(firstNames)
$entitySelection:=ds.Employee.all()
firstNames:=$entitySelection.firstName // el tipo de firstName es cadena
```

La colección resultante es una colección de cadenas, por ejemplo:

```
[ "Joanna", "Alexandra", "Rick" ]
```

Ejemplo 2

Proyección de entidad relacionada:

```
C_OBJECT($es;$entitySelection)
$entitySelection:=ds.Employee.all()
$es:=$entitySelection.employer // el empleador está relacionado con la clase de datos de Company
```

El objeto resultante es una selección de entidad de Company con duplicados eliminados (si los hay).

Ejemplo 3

Proyección de entidades relacionadas:

```
C_OBJECT($es)
$es:=ds.Employee.all().directReports // directReports <span id="result_box" lang="es"><span>está relacionado con</span></span>
Employee dataclass
```

El objeto resultante es una selección de entidad de Employee con duplicados eliminados (si los hay).

entitySelection.length

Parámetro	*Tipo*	*Descripción*
entitySelection.length	*Entero largo*	*Número de entidades en la entity selection*

Descripción

*La propiedad **entitySelection.length** devuelve el número de entidades en la entity selection. Si la entity selection está vacía, devuelve 0.*

*Las entity selections siempre tienen una propiedad **length**.*

Ejemplo

```
C_LONGINT(vSize)
vSize:=ds.Employee.query("gender = :1";"male").length
ALERT(String(vSize)+" male employees found.")
```

entitySelection.queryPath

Parámetro	*Tipo*	*Descripción*
entitySelection.queryPath	*Texto*	*Descripción de la consulta tal como se realizó en realidad*

Descripción

*La propiedad **entitySelection.queryPath** contiene una descripción detallada de la consulta como se realizó por 4D. Esta propiedad está disponible para objetos entitySelection generados mediante consultas si la propiedad "queryPath":true se pasó en el parámetro querySettings del método **query()**.*

*Para más información, consulte el párrafo **Parámetro confBusq**.*

entitySelection.queryPlan

Parámetro	*Tipo*	*Descripción*
entitySelection.queryPlan	*Texto*	*Descripción de la consulta justo antes de que se ejecute*

Descripción

*La propiedad **entitySelection.queryPlan** contiene una descripción detallada de la consulta justo antes de que se ejecute (es decir, la consulta planificada). Esta propiedad está disponible para objetos entitySelection generados mediante consultas si se pasó la propiedad "queryPlan":true en el parámetro querySettings del método **query()**.*

*Para más información, consulte el párrafo **Parámetro confBusq**.*

entitySelection[index]

Parámetro	*Tipo*	*Descripción*
entitySelection[index]	*Entity*	*Entidad correspondiente al índice especificado*

Descripción

*La notación **entitySelection[index]** le permite acceder a entidades dentro de la selección de entidad utilizando la sintaxis de colección estándar: pase la posición de la entidad que desea obtener en el parámetro indice.*

Tenga en cuenta que la entidad correspondiente se vuelve a cargar desde el almacén de datos.

*indice puede ser cualquier número entre 0 y **entitySelection.length**-1.*

- *Si indice está fuera de rango, se devuelve un error.*
- *Si indice corresponde a una entidad soltada, se devuelve un valor Null.*

***entitySelection[index]** es una **expresión no asignable**, lo que significa que no se puede usar como una referencia de entidad editable con métodos como **entity.lock( )** o **entity.save( )**. Para trabajar con la entidad correspondiente, debe asignar la expresión devuelta a una expresión asignable, como una variable. Ejemplos:*

```
$sel:=ds.Employee.all() //creación de la entity selection
 //invalid statements:
$result:=$sel[0].lock() //NO funcionará
$sel[0].lastName:="Smith" //NO funcionará
$result:=$sel[0].save() //NO funcionará
 //valid code:
$entity:=$sel[0]  //OK
$entity.lastName:="Smith" //OK
$entity.save() //OK
```

Ejemplo

```
C_OBJECT($employees;$employee)
$employees:=ds.Employee.query("lastName = :1";"H@")
$employee:=$employees[2]  // La tercera entidad de la selección de entidad $employees se vuelve a cargar desde la base
```

Create entity selection

Create entity selection (dsTable) -> Resultado

Parámetro	Tipo		Descripción
dsTable	Tabla	➡	Tabla en la base 4D cuya selección actual se usará para crear la selección de entidad
Resultado	EntitySelection	⮌	Selección de entidad que coincide con la clase de datos relacionada con la tabla dada

Descripción

*El comando **Create entity selection** crea y devuelve una nueva selección de entidad relacionada con la clase de datos que coincide con dsTable, de acuerdo con la selección actual de esta tabla.*

*Se crea una selección de entidad ordenada (se guarda el orden de la selección actual). Para más información, consulte el párrafo **Ordenadas vs No ordenadas** en la Guía del desarrollador 4D.*

*Si la dsTable no está expuesta en **ds** , se devuelve un error.*

Ejemplo

```
C_OBJECT($employees)
ALL RECORDS([Employee])
$employees:=Create entity selection([Employee])
  // La selección de la entidad $employees ahora contiene un conjunto de referencias sobre todas las entidades relacionadas con la clase de datos Employee
```

entitySelection.add( )

entitySelection.add (entidad)

Parámetro	Tipo		Descripción
entidad	Entity	➡	Entidad a añadir a la entity selection

Descripción

El método ***entitySelection.add( )*** *agrega la entidad especificada a la entity selection.*
Nota: *este método modifica la entity selection original.*

- *Si se ordena la entity selection, entidad se agrega al final de la selección. Si una referencia a la misma entidad ya pertenece a la entity selection, se duplica y se agrega una nueva referencia.*
- *Si la entity selection no está ordenada, entidad se agrega en cualquier parte de la selección, sin un orden específico.*

Nota: *para más información, consulte el párrafo* ***Ordenadas vs No ordenadas*** *en la Guía del desarrollador 4D.*
Se produce un error si la entidad y la entity selection no están relacionadas con la misma dataClass. Si la entidad a agregar es Null, no se genera ningún error.

Ejemplo

```
C_OBJECT($employees;$employee)
$employees:=ds.Employee.query("lastName = :1";"S@")
$employee:=ds.Employee.new()
$employee.lastName:="Smith"
$employee.save()
$employees.add($employee) //La entidad $employee se añade a la entity selection $employees
```

entitySelection.and( )

entitySelection.and (entidad | seleccionEntidad) -> Resultado

Parámetro	Tipo		Descripción
entidad | seleccionEntidad	Entity, EntitySelection	➡	Entidad o selección de entidad a interceptar con
Resultado	EntitySelection	⮌	Nueva selección de entidad con el resultado de intersección con AND lógico

Descripción

*El método **entitySelection.and( )** combina la selección de entidad con la entidad o seleccionEntidad utilizando el operador AND lógico; devuelve una nueva selección de entidad desordenada que contiene solo las entidades a las que se hace referencia tanto en la selección de entidad como en el parámetro.*

- *Si pasa entidad como parámetro, combina esta entidad con la selección de entidad. Si la entidad pertenece a la selección de entidad, se devuelve una nueva selección de entidad que contiene solo la entidad. De lo contrario, se devuelve una selección de entidad vacía.*
- *Si pasa seleccionEntidad como parámetro, combina ambas selecciones de entidad. Se devuelve una nueva selección de entidad que contiene solo las entidades a las que se hace referencia en ambas selecciones. Si no hay entidad que se cruza, se devuelve una selección de entidad vacía.*

***Nota:** puede comparar selecciones de entidades ordenadas y/o no ordenadas. La selección resultante siempre está desordenada. Para más información, consulte el párrafo **Ordenadas vs No ordenadas** en la Guía del desarrollador 4D.*

Si la selección de entidad original o el parámetro seleccionEntidad está vacío, o si la entidad es Null, se devuelve una selección de entidad vacía.
Si la selección de entidad original y el parámetro no están relacionados con la misma clase de datos, se genera un error.

Ejemplo 1

```
C_OBJECT($employees1;$employee;$result)
$employees1:=ds.Employee.query("lastName = :1";"H@") //La selección de entidad $employees1 contiene la entidad con la llave primaria 710 y otras entidades
 //por ej. "Colin Hetrick" / "Grady Harness" / "Sherlock Holmes" (llave primaria 710)
$employee:=ds.Employee.get(710) // Returns "Sherlock Holmes"

$result:=$employees1.and($employee) //$result es una selección de entidad que contiene solo la entidad con con llave primaria 710 ("Sherlock Holmes")
```

Ejemplo 2

Queremos tener una selección de empleados de nombre "Jones" que vivan en Nueva York:

```
C_OBJECT($sel1;$sel2;$sel3)
$sel1:=ds.Employee.query("name =:1";"Jones")
$sel2:=ds.Employee.query("city=:1";"New York")
$sel3:=$sel1.and($sel2)
```

entitySelection.average( )

entitySelection.average (rutaAtributo) -> Resultado

Parámetro	Tipo		Descripción
rutaAtributo	Texto	➡	Ruta de atributo que se utilizará para el cálculo
Resultado	Null, Real	➲	Media aritmética (promedio) de los valores de los atributos de la entidad

Descripción

*El método **entitySelection.average( )** devuelve la media aritmética (promedio) de todos los valores no nulos de rutaAtributo en la selección de entidad.*

Pase en el parámetro rutaAtributo la ruta del atributo a evaluar.

*Solo se utilizan los valores numéricos para el cálculo. Sin embargo, tenga en cuenta que, cuando rutaAtributo de la selección de entidades contiene tipos de valores mixtos, **entitySelection.average( )** tiene en cuenta todos los elementos para calcular el valor promedio.*

***Nota:** los valores de tipo fecha se convierten en valores numéricos (segundos) y se usan para calcular el promedio.*

***entitySelection.average( )** devuelve **null** si la entity selection está vacía:*

Se devuelve un error si:

- *rutaAtributo es un atributo relacionado o no contiene valores numéricos,*
- *rutaAtributo no se encuentra en la clase de datos de selección de entidad.*

Ejemplo

Queremos obtener la lista de empleados cuyo salario es más alto que el salario promedio:

```
C_REAL($averageSalary)
C_OBJECT($moreThanAv)
$averageSalary:=ds.Employee.all().average("salary")
$moreThanAv:=ds.Employee.query("salary > :1";$averageSalary)
```

entitySelection.contains( )

entitySelection.contains (entidad) -> Resultado

Parámetro	Tipo		Descripción
entidad	Entity	➡	Entidad a evaluar
Resultado	Booleano	⮌	True si la entidad pertenece a la selección de entidad, de lo contrario false

Descripción

*El método **entitySelection.contains( )** devuelve **true** si la referencia entidad pertenece a la selección de la entidad, y falso de lo contrario.*

En entidad, especifique la entidad a buscar en la selección de entidad. Si la entidad es Null, el método devolverá falso.

Si entidad y la selección de entidad no pertenecen a la misma clase de datos, se genera un error.

Ejemplo

```
C_OBJECT($employees;$employee)

$employees:=ds.Employee.query("lastName=:1";"H@")
$employee:=ds.Employee.get(610)

If($employees.contains($employee))
    ALERT("La entidad con la llave primaria 610 tiene un apellido que comienza por H")
Else
    ALERT("T<span id="result_box" lang="es"><span>La entidad con la clave primaria 610 no tiene un apellido que comienza con H</span></span>")
End if
```

entitySelection.count( )

entitySelection.count (rutaAtributo) -> Resultado

Parámetro	Tipo		Descripción
rutaAtributo	Texto	→	Ruta del atributo que se utilizará para el cálculo
Resultado	Real	←	Número de rutaAtributo no null en la selección de entidad

Descripción

El método ***entitySelection.count( )*** *devuelve el número de entidades en la selección de entidad con un valor no null en rutaAtributo.*

Nota: *solo se tienen en cuenta los valores escalares. Los valores de tipo objeto o colección se consideran valores nulos.*

Se devuelve un error si:

- *rutaAtributo es un atributo relacionado,*
- *rutaAtributo no se encuentra en la clase de datos de selección de entidad.*

Ejemplo

Queremos conocer el número total de empleados de una empresa sin contar los que cuyo nombre de puesto de trabajo no se haya especificado:

```
C_OBJECT($sel)
C_REAL($count)

$sel:=ds.Employee.query("employer = :1";"Acme, Inc")
$count:=$sel.count("jobname")
```

Y si quiere saber la cantidad de nombres de puestos de trabajo:

```
$countJobs:=$sel.count("jobname";dk distinct values)
```

entitySelection.distinct( )

entitySelection.distinct (rutaAtributo {; opcion}) -> Resultado

Parámetro	*Tipo*		*Descripción*
rutaAtributo	*Texto*	➡	*Ruta de atributo cuyos distintos valores desea obtener*
opcion	*Entero largo*	➡	*dk diacritical: evaluación diacrítica ("A" # "a" por ejemplo)*
Resultado	*Collection*	⬅	*Colección con valores diferentes únicamente*

Descripción

*El método **entitySelection.distinct( )** devuelve una colección que contiene solo valores distintos (diferentes) de rutaAtributo en la selección de entidad.*

*La colección devuelta se ordena automáticamente. Los valores **Null** no son devueltos.*

Pase en el parámetro rutaAtributo el atributo de entidad cuyos distintos valores desea obtener. Solo se pueden manejar valores escalares (texto, número, booleano o fecha). Si rutaAtributo es un atributo de objeto que contiene valores de diferentes tipos, primero se agrupan por tipo y se ordenan después. Los tipos se devuelven en el siguiente orden:

1. *booleanos*
2. *cadenas*
3. *números*
4. *fechas*

Por defecto, se realiza una evaluación no diacrítica. Si desea que la evaluación distinga entre mayúsculas y minúsculas o para diferenciar los caracteres acentuados, pase la constante dk diacritical en el parámetro opcion.

Se devuelve un error si:

- *rutaAtributo es un atributo relacionado,*
- *rutaAtributo o se encuentra en la clase de datos de selección de entidad.*

Ejemplo

Usted desea obtener una colección que contenga un solo elemento por nombre de país:

```
C_COLLECTION($countries)
$countries:=ds.Employee.all().distinct("address.country")
```

entitySelection.drop( )

entitySelection.drop ({modo}) -> Resultado

Parámetro	Tipo		Descripción
modo	Entero largo	➡	dk stop dropping on first error: detiene la ejecución del método en la primera entidad no soltable
Resultado	EntitySelection	➲	La selección de la entidad vacía es exitosa; de lo contrario, la selección de la entidad que contiene la(s) entidad(es) no soltable(s)

Descripción

*El método **entitySelection.drop( )** elimina las entidades que pertenecen a la selección de entidad de la tabla relacionada con su clase de datos dentro del almacén de datos. La selección de entidad permanece en la memoria.*

***Nota**: la eliminación de entidades es permanente y no se puede deshacer. Se recomienda llamar a esta acción en una transacción para tener una opción de reversión.*

*Si se encuentra una entidad bloqueada durante la ejecución de **entitySelection.drop( )**, no se elimina. Por defecto, el método procesa todas las entidades de la selección de entidad y devuelve entidades no soltables en la selección de entidad. Si desea que el método detenga la ejecución en la primera entidad no soltable encontrada, pase la constante dk stop dropping on first error en el parámetro modo.*

Ejemplo

```
C_TEXT(idsNotDeletedEntities)
C_OBJECT(vNotDroppedES)
vNotDroppedES:=Form.empsToBeDeleted.drop()  //Form.empsToBeDeleted es una selección de entidad a soltar
If(vNotDroppedES.length=0)
    ALERT("Drop done for "+String(Form.empsToBeDeleted.length)+" employees")
    Form.empsToBeDeleted:=ds.Employee.newSelection(dk keep ordered)
Else
    For each($emp;vNotDroppedES)
        idsNotDeletedEntities:=idsNotDeletedEntities+String($emp.getKey(dk key as string))+" "
 // Crea una lista de llaves primarias de entidades no soltadas
    End for each
End if
```

entitySelection.first( )

entitySelection.first () -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entity, Null*	➲	*Referencia a la primera entidad de la selección de entidad*

Descripción

*El método **entitySelection.first( )** devuelve una referencia a la entidad en la primera posición de la selección de entidad.*
El resultado de este método es similar a:

```
$entity:=$entitySel[0]
```

Sin embargo, hay una diferencia entre ambas declaraciones cuando la selección está vacía:

```
C_OBJECT($entitySel;$entity)
$entitySel:=ds.Emp.query("lastName = :1";"Nonexistentname") //no hay entidad coincidente
 //la selección de entidad está vacía
$entity:=$entitySel.first() //devuelve Null
$entity:=$entitySel[0]  //genera un error
```

Ejemplo

```
C_OBJECT($entitySelection;$entity)
$entitySelection:=ds.Emp.query("salary > :1";100000)
If($entitySelection.length#0)
    $entity:=$entitySelection.first()
End if
```

entitySelection.isOrdered( )

entitySelection.isOrdered () -> Resultado

Parámetro	*Tipo*	*Descripción*
Resultado	*Booleano*	*True si la entity selection está ordenada, de lo contrario false*

Descripción

*El método **entitySelection.isOrdered( )** devuelve true si la entity selection está ordenada, y false si está desordenada.*
***Nota**: para más información, consulte el párrafo **Ordenadas vs No ordenadas**.*

Ejemplo

```
C_OBJECT($employees;$employee)
C_BOOLEAN($isOrdered)
$employees:=ds.Employee.newSelection(dk keep ordered)
$employee:=ds.Employee.get(714)  // Obtiene la entidad con la llave primaria 714

 //En una entity selection ordenada, podemos agregar varias veces la misma entidad (se guardan los duplicados)
$employees.add($employee)
$employees.add($employee)
$employees.add($employee)

$isOrdered:=$employees.isOrdered()
If($isOrdered)
    ALERT("The entity selection is ordered and contains "+String($employees.length)+" employees")
End if
```

entitySelection.last( )

entitySelection.last () -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entity, Null*		*Referencia a la última entidad de la selección de la entidad*

Descripción

*El método **entitySelection.last( )** devuelve una referencia a la entidad en la última posición de la selección de entidad.*
El resultado de este método es similar a:

```
$entity:=$entitySel[length-1]
```

Si la selección de la entidad está vacía, el método devuelve Null.

Ejemplo

```
C_OBJECT($entitySelection;$entity)
$entitySelection:=ds.Emp.query("salary < :1";50000)
If($entitySelection.length#0)
    $entity:=$entitySelection.last()
End if
```

entitySelection.max( )

entitySelection.max (rutaAtributo) -> Resultado

Parámetro	Tipo		Descripción
rutaAtributo	Texto	→	Ruta del atributo que se utilizará para el cálculo
Resultado	Mixed	←	Valor más alto de atributo

Descripción

El método ***entitySelection.max( )*** *devuelve el valor más alto (o máximo) entre todos los valores de rutaAtributo en la selección de entidades. De hecho, devuelve el valor de la última entidad de la selección de entidades, ya que se ordenaría en orden ascendente utilizando el método* ***entitySelection.orderBy( )****.*

Si pasó en rutaAtributo una ruta a un atributo de objeto que contiene diferentes tipos de valores, el método ***entitySelection.max( )*** *devolverá el valor máximo dentro del primer tipo de valor escalar en el orden por defecto de la lista de los tipos 4D (ver* ***collection.sort( )****). En este caso, si rutaAtributo no existe en el objeto,* ***entitySelection.max( )*** *devuelve* ***null****.*

Si la selección de entidades está vacía, ***entitySelection.max( )*** *devuelve* ***null****.*

Se devuelve un error si:

- *rutaAtributo es un atributo relacionado,*
- *rutaAtributo no se encuentra en la selección de entidades de la clase de datos.*

Ejemplo

Queremos encontrar el salario más alto entre todas las empleadas:

```
C_OBJECT($sel)
C_REAL($maxSalary)
$sel:=ds.Employee.query("gender = :1";"female")
$maxSalary:=$sel.max("salary")
```

entitySelection.min( )

entitySelection.min (rutaAtributo) -> Resultado

Parámetro	Tipo		Descripción
rutaAtributo	Texto	→	Ruta del atributo que se utilizará para el cálculo
Resultado	Mixed	←	Valor más bajo del atributo

Descripción

El método ***entitySelection.min( )*** *devuelve el valor más bajo (o mínimo) entre todos los valores de rutaAtributo en la selección de entidad. En realidad, devuelve la primera entidad de la selección de entidad, ya que se ordenaría en orden ascendente utilizando el método* ***entitySelection.orderBy( )****.*

Si pasa en rutaAtributo una ruta a un atributo de objeto que contiene diferentes tipos de valores, el método ***entitySelection.min( )*** *devolverá el valor mínimo del primer tipo de valor escalar en el orden estándar de los tipos (ver la descripción del método* ***collection.sort( )****). En este caso, si rutaAtributo no existe en el objeto,* ***entitySelection.min( )*** *devuelve* ***null****.*

Si la selección de la entidad está vacía, ***entitySelection.min( )*** *devuelve* ***null****.*

Se devuelve un error si:

- *rutaAtributo es un atributo relacional,*
- *rutaAtributo no se encuentra en la selección de entidades de la clase de datos.*

Ejemplo

En este ejemplo, queremos encontrar el salario más bajo entre todas las empleadas:

```
C_OBJECT($sel)
C_REAL($minSalary)
$sel:=ds.Employee.query("gender = :1";"female")
$minSalary:=$sel.min("salary")
```

entitySelection.minus( )

entitySelection.minus (entidad | seleccionEntidad) -> Resultado

Parámetro	Tipo		Descripción
entidad | seleccionEntidad	Entity, EntitySelection	➡	Selección de entidad o entidad a restar
Resultado	EntitySelection	⮌	Nueva selección de entidad o una nueva referencia en la selección de entidad existente

Descripción

*El método **entitySelection.minus( )** excluye de la selección de entidad a la que se aplica la entidad o las entidades de seleccionEntidad y devuelve la selección de entidad resultante.*

- *Si pasa entidad como parámetro, el método crea una nueva selección de entidad sin entidad (si entidad pertenece a la selección de entidad). Si entidad no se incluyó en la selección de la entidad original, se devuelve una nueva referencia a la selección de la entidad.*
- *Si pasa seleccionEntidad como parámetro, el método devuelve una selección de entidad que contiene las entidades que pertenecen a la selección de entidad original de la que se han eliminado las entidades que pertenecen a seleccionEntidad.*

***Nota:** puede comparar selecciones de entidades ordenadas y/o no ordenadas. La selección resultante siempre está desordenada. Para más información, consulte el párrafo **Ordenadas vs No ordenadas** en la Guía del desarrollador 4D.*

Si la selección de entidad original o ambos la selección de entidad original y el parámetro seleccionEntidad están vacíos, se devuelve una selección de entidad vacía.
Si seleccionEntidad está vacía o si entidad es Null, se devuelve una nueva referencia a la selección de la entidad original.

Si la selección de entidad original y el parámetro no están relacionados con la misma clase de datos, se genera un error.

Ejemplo 1

```
C_OBJECT($employees;$employee;$result)

$employees:=ds.Employee.query("lastName = :1";"H@") // La selección de entidad $employees contiene la entidad con la llave primaria 710 y otras entidades
  // por ej. "Colin Hetrick", "Grady Harness", "Sherlock Holmes" (primary key 710)

$employee:=ds.Employee.get(710) // Devuelve "Sherlock Holmes"

$result:=$employees.minus($employee) //$result contiene "Colin Hetrick", "Grady Harness"
```

Ejemplo 2

Queremos tener una selección de empleadas llamadas "Jones" que vivan en Nueva York:

```
C_OBJECT($sel1;$sel2;$sel3)
$sel1:=ds.Employee.query("name =:1";"Jones")
$sel2:=ds.Employee.query("city=:1";"New York")
$sel3:=$sel1.and($sel2).minus(ds.Employee.query("gender='male'"))
```

entitySelection.or( )

entitySelection.or (entidad | entitySelection) -> Resultado

Parámetro	Tipo		Descripción
entidad | entitySelection	Entity, EntitySelection	➡	Entidad o entity selection a intersectar con
Resultado	EntitySelection	⮌	Nueva entity selection o nueva referencia a la entity selection original

Descripción

El método ***entitySelection.or( )*** *combina la entity selection con el parámetro entidad o entitySelection utilizando el operador lógico (no exclusivo) OR; devuelve una nueva entity selection desordenada que contiene todas las entidades de la entity selection y el parámetro.*

- *Si pasa entidad como parámetro, compara esta entidad con la entity selection. Si la entidad pertenece a la entity selection, se devuelve una nueva referencia a la entity selection. De lo contrario, se devuelve una nueva entity selection que contiene la entity selection original y la entidad.*
- *Si pasa entitySelection como parámetro, compara las entity selections. Se devuelve una nueva entity selection que contiene las entidades que pertenecen a la entity selection original o entitySelection (o no es exclusiva, las entidades a las que se hace referencia en ambas selecciones no se duplican en la selección resultante).*

Nota: *puede comparar entity selections ordenadas y/o no ordenadas. La selección resultante siempre está desordenada. Para más información, consulte el párrafo* ***Ordenadas vs No ordenadas*** *en la Guía del desarrollador 4D.*

Si la entity selection original y el parámetro entitySelection están vacíos, se devuelve una entity selection vacía. Si la entity selection original está vacía, se devuelve una referencia a entitySelection o se devuelve una entity selection que contiene solo la entidad.

Si entitySelection está vacía o si entidad es Null, se devuelve una nueva referencia a la entity selection original.

Si la entity selection original y el parámetro no están relacionados con la misma dataclass, se genera un error.

Ejemplo 1

```
C_OBJECT($employees1;$employees2;$result)
$employees1:=ds.Employee.query("lastName = :1";"H@") //Devuelve "Colin Hetrick","Grady Harness"
$employees2:=ds.Employee.query("firstName = :1";"C@") //Devuelve "Colin Hetrick", "Cath Kidston"
$result:=$employees1.or($employees2) //$result contiene "Colin Hetrick", "Grady Harness","Cath Kidston"
```

Ejemplo 2

```
C_OBJECT($employees;$employee;$result)
$employees:=ds.Employee.query("lastName = :1";"H@") // Devuelve "Colin Hetrick","Grady Harness", "Sherlock Holmes"
$employee:=ds.Employee.get(686) //la entidad con llave primaria 686 no pertenece a la entity selection $employees
 //Coincide el empleado "Mary Smith"

$result:=$employees.or($employee) //$result contiene "Colin Hetrick", "Grady Harness", "Sherlock Holmes", "Mary Smith"
```

entitySelection.orderBy( )

entitySelection.orderBy (criteria) -> Resultado

Parámetro	*Tipo*		*Descripción*
criteria	*Texto, Collection*	➡	*Texto: ruta(s) de atributo y orden para ordenar la selección de entidad* *Colección: colección de objetos de criterio*
Resultado	*EntitySelection*	⮌	*Nueva entity selection en el orden especificado*

Descripción

*El método **entitySelection.orderBy( )** devuelve una nueva entity selection ordenada que contiene todas las entidades de la entity selection en el orden especificado por criteria.*

Notas:

- *Este método no modifica la selección de la entidad original.*
- *Para más información sobre las entity selections ordenadas, consulte **Ordenadas vs No ordenadas**.*

Debe utilizar el parámetro criteria para definir cómo deben ordenarse las entidades. Dos sintaxis son soportadas para este parámetro:

- *criteria es del **tipo texto** (formula): en este caso, criteria contiene una fórmula compuesta de 1 a x rutas de atributos y (opcionalmente) órdenes de clasificación, separadas por comas. La sintaxis de la fórmula es:*

  ```
  "attributePath1 {desc or asc}, attributePath2 {desc or asc},..."
  ```

  El orden en que se pasan los atributos determina la prioridad de ordenación de las entidades. Por defecto, los atributos se ordenan en orden ascendente. Puede establecer el orden de clasificación de una propiedad en la cadena de criterios, separada de la ruta de la propiedad por un espacio único: pase "asc" para ordenar en orden ascendente o "desc" en orden descendente.

- *criteria es del **tipo colección**: en este caso, cada elemento de la colección contiene un objeto estructurado de la siguiente manera:*
  {
  "propertyPath": cadena,
  "descending": booleano
  }
  Por defecto, los atributos se ordenan en orden ascendente ("descendente" es false).
  Puede agregar tantos objetos en la colección criteria como sea necesario.

***Nota:** los valores Null se evalúan como valores menores que los otros valores.*

Ejemplo

```
// ordenar por fórmula
$sortedEntitySelection:=$entitySelection.orderBy("firstName asc, salary desc")
$sortedEntitySelection:=$entitySelection.orderBy("firstName")

// ordenar por colección con o sin ordenación
$orderColl:=New collection
$orderColl.push(New object("propertyPath";"firstName";"descending";False))
$orderColl.push(New object("propertyPath";"salary";"descending";True))
$sortedEntitySelection:=$entitySelection.orderBy($orderColl)

$orderColl:=New collection
$orderColl.push(New object("propertyPath";"manager.lastName"))
$orderColl.push(New object("propertyPath";"salary"))
$sortedEntitySelection:=$entitySelection.orderBy($orderColl)
```

entitySelection.query( )

entitySelection.query (cadenaBusq {; valor}{; valor2 ; ... ; valorN}{; opcionesBusq}) -> Resultado

Parámetro	Tipo		Descripción
cadenaBusq	Texto	➡	Criterio de búsqueda
valor		➡	Valor(es) a comparar al utilizar separadores
opcionesBusq	Objeto	➡	Opciones de búsqueda
Resultado	EntitySelection	⮌	Nueva entitySelection hecha de entidades de entity selections que cumplen con el criterio de búsqueda especificado en cadenaBusq

Descripción

*El método **entitySelection.query( )** busca entidades que cumplan los criterios de búsqueda especificados en cadenaBusq y valor (opcionalmente), para todas las entidades en dataClass o entitySelection, y devuelve un nuevo objeto de tipo EntitySelection que contiene todas las entidades de dataClass que son encontraron. Se aplica carga diferida.*

***Nota:** este método no modifica la selección de entidad original.*

Si no se encuentran entidades coincidentes, se devuelve una EntitySelection vacía.

*Para información detallada sobre cómo crear una consulta utilizando los parámetros cadenaBusq, valor, y opcionesBusq, consulte la descripción del método **dataClass.query( )**.*

***Nota:** la selección de entidad devuelta no está ordenada (para más información, consulte **Ordenadas vs No ordenadas**). Sin embargo, tenga en cuenta que, en el modo Cliente/Servidor, se comporta como una entity selection ordenada (las entidades se agregan al final de la selección).*

Ejemplo 1

```
C_OBJECT($entitySelectionTemp)
$entitySelectionTemp:=dataClass.query("lastName = :1";"M@")
Form.emps:=$entitySelectionTemp.query("manager.lastName = :1";"S@")
```

Ejemplo 2

*Se pueden encontrar más ejemplos de búsquedas en la página **dataClass.query( )**.*

entitySelection.slice( )

entitySelection.slice (iniciarDesde {; fin}) -> Resultado

Parámetro	Tipo		Descripción
iniciarDesde	Entero largo	➡	Índice para comenzar la búsqueda (incluido)
fin	Entero largo	➡	Índice final (no incluido)
Resultado	EntitySelection	⮌	Nueva selección de entidad que contiene entidades cortadas (copia superficial)

Descripción

*El método **entitySelection.slice( )** devuelve una parte de selección de entidades en una nueva selección de entidades, seleccionada desde el indice iniciarDesde hasta el indice fin (fin no incluido). Este método devuelve una copia superficial de la selección de entidades (se utilizan las mismas referencias de entidades).*

***Nota:** este método no modifica la selección de entidades original.*

La selección de entidades devuelta contiene las entidades especificadas por iniciarDesde y todas las entidades posteriores hasta, pero sin incluir, la entidad especificada por fin. Si solo se especifica el parámetro iniciarDesde, la selección de entidades devuelta contiene todas las entidades desde iniciarDesde hasta la última entidad de la selección de entidades original.

- *Si iniciarDesde < 0, se vuelve a calcular como iniciarDesde :=iniciarDesde +length (se considera como el desplazamiento desde el final de la selección de entidades). Si el valor calculado es < 0, iniciarDesde se establece en 0.*
- *Si iniciarDesde >= length, el método devuelve una selección de entidades vacía.*
- *Si fin < 0, se vuelve a calcular como fin:=fin +length.*
- *Si fin < iniciarDesde (valores pasados o calculados), el método no hace nada.*

Si la selección de entidades contiene entidades eliminadas, se devuelven.

Ejemplo 1

Usted desea obtener una selección de las 10 primeras entidades de la selección de una entidad:

```
C_OBJECT($sel;$sliced)
$sel:=ds.Employee.query("salary > :1";50000)
$sliced:=$sel.slice(0;9)
```

Ejemplo 2

Suponiendo que tenemos ds.Employee.all().length = 10

```
C_OBJECT($slice)
$slice:=ds.Employee.all().slice(-1;-2) //intenta devolver las entidades de posición 9 a 8, pero desde 9 > 8, devuelve una selección de entidades vacía
```

entitySelection.sum( )

entitySelection.sum (rutaAtributo) -> Resultado

Parámetro	Tipo		Descripción
rutaAtributo	Texto	→	Ruta del atributo que se utilizará para el cálculo
Resultado	Real	←	Suma de valores de selección de entidad

Descripción

El método ***entitySelection.sum( )*** *devuelve la suma de todos los valores rutaAtributo en la selección de entidad.*

entitySelection.sum( ) *devuelve 0 si la selección de entidad está vacía.*

La suma solo se puede hacer en valores del tipo de número. Si el tipo rutaAtributo es un objeto, solo se tienen en cuenta los valores numéricos para el cálculo (se ignoran otros tipos de valor). En este caso, si rutaAtributo conduce a una propiedad que no existe en el objeto o que no contiene ningún valor numérico, ***entitySelection.sum( )*** *devuelve 0.*

Se devuelve un error si:

- *rutaAtributo no es un atributo numérico o de objeto,*
- *rutaAtributo es un atributo relacionado,*
- *rutaAtributo no se encuentra en la clase de datos de selección de entidad.*

Ejemplo

```
C_OBJECT($sel)
C_REAL($sum)

$sel:=ds.Employee.query("salary < :1";20000)
$sum:=$sel.sum("salary")
```

entitySelection.toCollection( )

entitySelection.toCollection ({filtro ;}{ opciones {; inicio {; cuantas}}}) -> Resultado

Parámetro	*Tipo*		*Descripción*
filtro	*Cadena, Collection*	➡	*Especifica qué propiedades de entidad extraer*
opciones	*Entero largo*	➡	*dk with primary key: añade la llave primaria dk with stamp: añade el sello*
inicio	*Entero largo*	➡	*Designa el índice de inicio*
cuantas	*Entero largo*	➡	*Número de entidades a extraer*
Resultado	*Collection*	⮌	*Colección de objetos que contienen atributos y valores de la colección de entidades*

Descripción

*El método **entitySelection.toCollection( )** crea y devuelve una colección donde cada elemento es un objeto que contiene un conjunto de propiedades y valores correspondientes a los nombres y valores de atributos para la colección de entidades .*

Si se omite el parámetro filtro o contiene una cadena vacía o "", se extraen todos los atributos. Los atributos con la propiedad "kind" "relatedEntity" se extraen de la forma simple: un objeto con propiedad __KEY (llave principal). Los atributos con la propiedad "kind" "relatedEntities" no se extraen.*

En el parámetro filtro, puede pasar los atributos de entidad a extraer. Se permiten dos sintaxis:

- *una cadena con rutas de propiedad separadas por comas: "propertyPath1, propertyPath2, ...".*
- *una colección de cadenas: ["propertyPath1","propertyPath2";...]*

*Si filtro se especifica para atributos del tipo **relatedEntity**:*

- *propertyPath = "relatedEntity" -> se extrae de forma simple*
- *propertyPath = "relatedEntity.*" -> se extraen todas las propiedades*
- *propertyPath = "relatedEntity.propertyName1; relatedEntity.propertyName2; ..." -> solo se extraen esas propiedades*

*Si filtro se especifica para atributos del tipo **relatedEntities**:*

- *propertyPath = "relatedEntities.*" -> todas las propiedades se extraen*
- *propertyPath = "relatedEntities.propertyName1; relatedEntities.propertyName2; ..." -> solo se extraen esas propiedades*

En el parámetro opciones, puede pasar los selectores dk with primary key y/o dk with stamp para agregar las llaves principales y/o sellos de la entidad en los objetos extraídos.

El parámetro inicio le permite indicar el índice inicial de las entidades a extraer. Puede pasar cualquier valor entre 0 y la entidad de selección-1.

El parámetro cuantas le permite especificar el número de entidades a extraer, comenzando con la especificada en inicio. Las entidades soltadas no se devuelven, pero se tienen en cuenta según cuantas. Por ejemplo, si cuantas= 3 y hay 1 entidad soltada, solo se extraen 2 entidades.

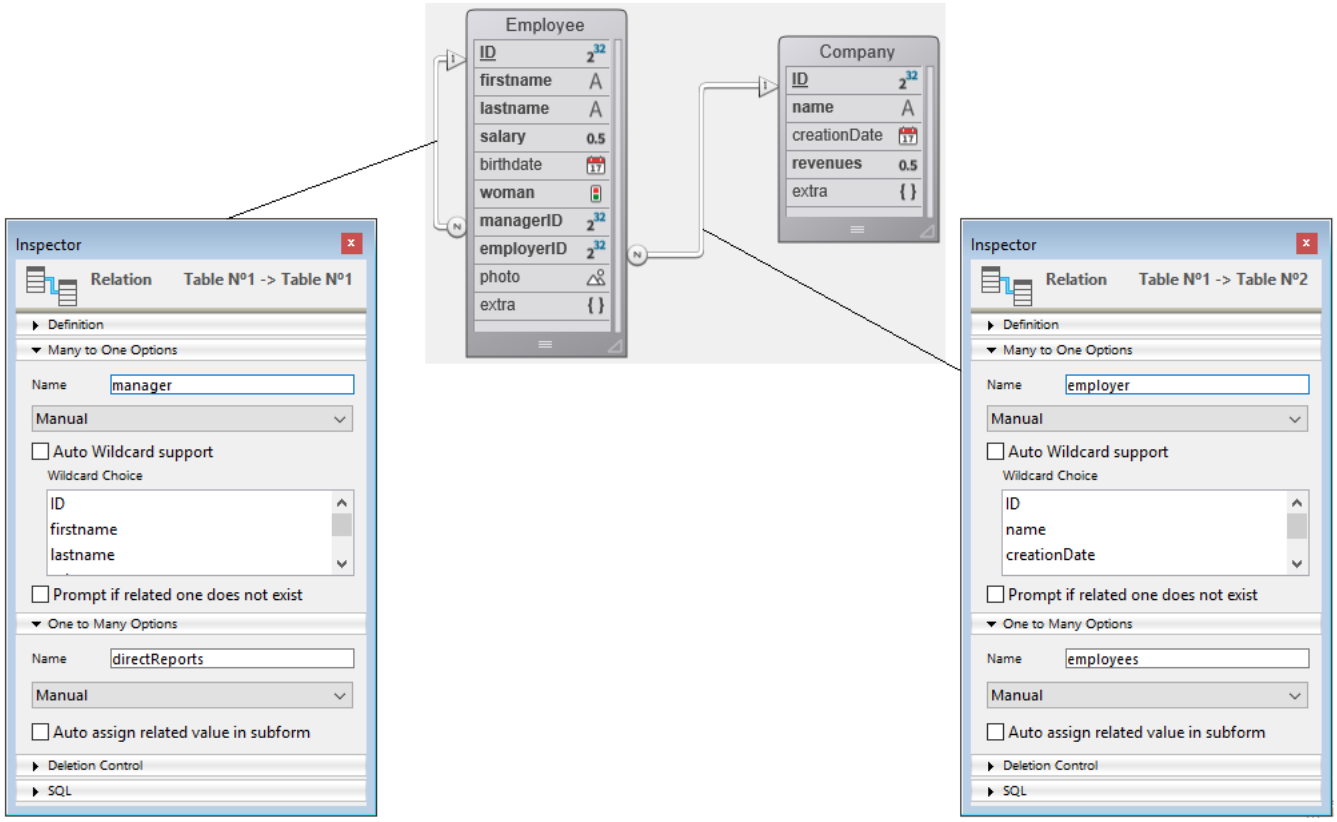
Si cuantas> longitud de la selección de entidad, el método devuelve objetos (length - inicio).

Se devuelve una colección vacía si:

- *la selección de entidad está vacía, o*
- *inicio es mayor que la longitud de la selección de entidad.*

Ejemplo 1

La siguiente estructura se usará en todos los ejemplos de esta sección:



Ejemplo sin parámetro filtro u opciones:

```
C_COLLECTION($employeesCollection)
C_OBJECT($employees)

$employeesCollection:=New collection
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection()
```

Devuelve:


```
[ { "ID": 416, "firstName": "Gregg", "lastName": "Wahl", "salary": 79100, "birthDate": "1963-02-01T00:00:00.000Z", "woman": false, "managerID": 412, "employerID": 20, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 20 }, "manager": { "__KEY": 412 } }, { "ID": 417, "firstName": "Irma", "lastName": "Durham", "salary": 47000, "birthDate": "1992-06-16T00:00:00.000Z", "woman": true, "managerID": 412, "employerID": 20, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 20 }, "manager": { "__KEY": 412 } } ]
```


Ejemplo 2

Ejemplo con opciones:

```
$employeesCollection:=New collection
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection("";dk with primary key+dk with stamp)
```

Devuelve:


```
[ { "__KEY": 416, "__STAMP": 1, "ID": 416, "firstName": "Gregg", "lastName": "Wahl", "salary": 79100, "birthDate": "1963-02-01T00:00:00.000Z", "woman": false, "managerID": 412, "employerID": 20, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 20 }, "manager": { "__KEY": 412 } }, { "__KEY": 417, "__STAMP": 1, "ID": 417, "firstName": "Irma", "lastName": "Durham", "salary": 47000, "birthDate": "1992-06-16T00:00:00.000Z", "woman": true, "managerID": 412, "employerID": 20, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 20 }, "manager": { "__KEY": 412 } } ]
```


Ejemplo 3

Ejemplo con découpage y filtro:

```
$employeesCollection:=Creer collection
$filter:=Creer collection
$filter.push("firstName")
$filter.push("lastName")
```

```
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection($filter;0;0;2)
```

Devuelve:

```
[ { "firstName": "Gregg", "lastName": "Wahl" }, { "firstName": "Irma", "lastName": "Durham" } ]
```

Ejemplo 4

*Ejemplo con el tipo **relatedEntity** con formulario simple:*

```
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName,lastName,employer")
```

Devuelve:

```
[ { "firstName": "Gregg", "lastName": "Wahl", "employer": { "__KEY": 20 } }, { "firstName": "Irma",
"lastName": "Durham", "employer": { "__KEY": 20 } }, { "firstName": "Lorena", "lastName": "Boothe",
"employer": { "__KEY": 20 } } ]
```

Ejemplo 5

Ejemplo con filtro como una colección:

```
$employeesCollection:=New collection
$coll:=New collection("firstName";"lastName")
$employeesCollection:=$employees.toCollection($coll)
```

Devuelve:

```
[ { "firstName": "Joanna", "lastName": "Cabrera" }, { "firstName": "Alexandra", "lastName": "Coleman" } ]
```

Ejemplo 6

Ejemplo con extracción de todas las propiedades de una relatedEntity:

```
$employeesCollection:=New collection
$coll:=New collection
$coll.push("firstName")
$coll.push("lastName")
$coll.push("employer.*")
$employeesCollection:=$employees.toCollection($coll)
```

Devuelve:

```
[ { "firstName": "Gregg", "lastName": "Wahl", "employer": { "ID": 20, "name": "India Astral Secretary",
"creationDate": "1984-08-25T00:00:00.000Z", "revenues": 12000000, "extra": null } }, { "firstName": "Irma",
"lastName": "Durham", "employer": { "ID": 20, "name": "India Astral Secretary", "creationDate": "1984-08-
25T00:00:00.000Z", "revenues": 12000000, "extra": null } }, { "firstName": "Lorena", "lastName": "Boothe",
"employer": { "ID": 20, "name": "India Astral Secretary", "creationDate": "1984-08-25T00:00:00.000Z",
"revenues": 12000000, "extra": null } } ]
```

Ejemplo 7

Ejemplo con extracción de algunas propiedades de una relatedEntity:

```
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName, employer.name")
```

```
[ { "firstName": "Gregg", "lastName": "Wahl", "employer": { "name": "India Astral Secretary" } }, {
"firstName": "Irma", "lastName": "Durham", "employer": { "name": "India Astral Secretary" } }, { "firstName":
"Lorena", "lastName": "Boothe", "employer": { "name": "India Astral Secretary" } }]
```

Ejemplo 8

*Ejemplo con extracción de algunas propiedades de **relatedEntities**:*

```
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName, directReports.firstName")
```

Devuelve:

```
[ { "firstName": "Gregg", "lastName": "Wahl", "directReports": [] }, { "firstName": "Mike", "lastName": "Phan",
"directReports": [ { "firstName": "Gary" }, { "firstName": "Sadie" },
```

{ "firstName": "Christie" }] }, { "firstName": "Gary", "lastName": "Reichert", "directReports": [{ "firstName": "Rex" }, { "firstName": "Jenny" }, { "firstName": "Lowell" }] }]

Ejemplo 9

*Ejemplo con extracción de todas las propiedades de **relatedEntities**:*

```
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName, directReports.*")
```

[{ "firstName": "Gregg", "lastName": "Wahl", "directReports": [] }, { "firstName": "Mike", "lastName": "Phan", "directReports": [{ "ID": 425, "firstName": "Gary", "lastName": "Reichert", "salary": 65800, "birthDate": "1957-12-23T00:00:00.000Z", "woman": false, "managerID": 424, "employerID": 21, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 21 }, "manager": { "__KEY": 424 } }, { "ID": 426, "firstName": "Sadie", "lastName": "Gallant", "salary": 35200, "birthDate": "2022-01-03T00:00:00.000Z", "woman": true, "managerID": 424, "employerID": 21, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 21 }, "manager": { "__KEY": 424 } }] }, { "firstName": "Gary", "lastName": "Reichert", "directReports": [{ "ID": 428, "firstName": "Rex", "lastName": "Chance", "salary": 71600, "birthDate": "1968-08-09T00:00:00.000Z", "woman": false, "managerID": 425, "employerID": 21, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 21 }, "manager": { "__KEY": 425 } }, { "ID": 429, "firstName": "Jenny", "lastName": "Parks", "salary": 51300, "birthDate": "1984-05-25T00:00:00.000Z", "woman": true, "managerID": 425, "employerID": 21, "photo": "[object Picture]", "extra": null, "employer": { "__KEY": 21 }, "manager": { "__KEY": 425 } }] }]

USE ENTITY SELECTION

USE ENTITY SELECTION (entitySelection)

Parámetro	Tipo		Descripción
entitySelection	EntitySelection	➡	Una selección de entidad

Descripción

*El comando **USE ENTITY SELECTION** actualiza la selección actual de la tabla que coincide con la clase de datos del parámetro entitySelection, de acuerdo con el contenido de la selección de entidades.*

*Este comando solo funciona con el almacén de datos local o cliente/servidor devuelto por **ds** .*

***Nota:** después de una llamada a **USE ENTITY SELECTION**,el primer registro de la selección actual actualizada (si no está vacío) se convierte en el registro actual, pero no se carga en memoria. Si necesita utilizar los valores de los campos en el registro actual, utilice el comando **LOAD RECORD** cdespués del comando **USE ENTITY SELECTION**.*

Ejemplo

```
C_OBJECT($entitySel)
$entitySel:=ds.Employee.query("lastName = :1";"M@") //$entitySel está relacionado con la clase de datos Employee
REDUCE SELECTION([Employee];0)
USE ENTITY SELECTION($entitySel) //Se actualiza la selección actual de la tabla Employee
```

PHP

Ejecutar scripts PHP en 4D
PHP Execute
PHP GET FULL RESPONSE
PHP GET OPTION
PHP SET OPTION
Soporte de módulos PHP

Ejecutar scripts PHP en 4D

4D v12 permite ejecutar directamente los scripts PHP. Esta nueva posibilidad da acceso a valiosas librerías utilitarias disponibles vía PHP. Estas librerías ofrecen en particular funciones de:

- *cifrado (MD5) y hashing,*
- *manipulación de archivos ZIP,*
- *manipulación de imágenes,*
- *acceso LDAP,*
- *acceso COM (control de documentos MS Office), etc.*

Esta lista no es exhaustiva. Para una lista completa de los módulos PHP disponibles por defecto con 4D, consulte la sección ***Soporte de módulos PHP****. Note también que es posible instalar módulos personalizados adicionales.*

Para ejecutar un script o una función PHP, debe utilizar el comando ***PHP Execute****. Por defecto, 4D incluye la versión 5.4.11 de PHP. Los scripts ejecutados deben ser compatibles con esta versión y con los módulos instalados.*
Para una descripción completa de los comandos y de la sintaxis PHP, consulte la extensa documentación PHP disponible en Internet. Como ejemplo, estas son las direcciones de algunos sitios de referencia:
http://us.php.net/manual/en/
http://phpdeveloper.org/
http://php.start4all.com/

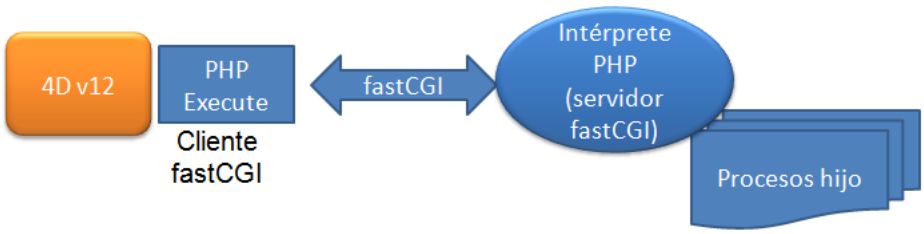
Arquitectura

4D ofrece un intérprete PHP compilado en FastCGI, protocolo de comunicación de tipo cliente servidor entre una aplicación y un intérprete PHP.

El intérprete PHP dirige un conjunto de procesos de ejecución llamado "procesos hijos". Estos procesos están dedicados al procesamiento de las peticiones enviadas por 4D. La ejecución de las peticiones es sincrónica. Por razones de optimización, por defecto hasta cinco procesos hijos pueden correr simultáneamente (este número puede modificarse vía las Propiedades de la base o el comando ***SET DATABASE PARAMETER****). Bajo Mac OS, estos procesos se lanzan a la primera petición y son conservados permanentemente por el intérprete PHP. Bajo Windows, 4D crea los procesos en función de las necesidades y los recicla si es necesario. 4D soporta automáticamente la gestión de los procesos del intérprete PHP ofrecida por defecto (lanzamiento y cierre).*

Nota: *si el programa 4D cierra inesperadamente mientras los procesos PHP hijos aún están activos, debe borrarlos manualmente vía la ventana de gestión de procesos del sistema.*

El siguiente diagrama ilustra la arquitectura 4D/PHP de 4D:



Esta arquitectura funciona con un sistema de peticiones internas enviadas por 4D a una dirección TCP predefinida (dirección IP y número de puerto). Si es necesario, por ejemplo si varios intérpretes PHP se ejecutan en la misma máquina, esta dirección puede modificarse vía las Propiedades de la base o vía el comando ***SET DATABASE PARAMETER****.*

Atención: *si lanza dos aplicaciones 4D en la misma máquina y ejecuta instrucciones PHP en cada una de ellas (vía el comando* ***PHP Execute****), es imperativo modificar los puertos receptores del intérprete FastCGI PHP con el fin de utilizar uno diferente para cada aplicación. De lo contrario, la ejecución de las instrucciones PHP puede fallar de manera aleatoria e incluso bloquear la aplicación 4D.*

Utilizar otro intérprete PHP y otro archivo php.ini

Puede elegir utilizar otro intérprete PHP diferente al ofrecido por 4D. Esto le permite conservar un mismo intérprete PHP incluso si actualiza 4D. Además, le permite instalar todos los módulos personalizados que desee, de hecho, no se puede utilizar un archivo php.ini personalizado con el intérprete incluido en 4D. Para utilizar opciones de configuración de php diferentes a las que se ofrecen por defecto, debe administrar un intérprete externo.

Un intérprete PHP personalizado debe respetar dos condiciones:

- *Debe compilarse en FastCGI,*
- *Debe estar en la misma máquina que 4D.*

Para utilizar un intérprete PHP personalizado, simplemente debe configurarlo de manera que escuche una dirección y un puerto TCP específicos e indique a 4D no activar el intérprete interno. Estos parámetros pueden especificarse vía las Propiedades de la base o para la sesión vía el comando ***SET DATABASE PARAMETER****. En este caso, debe administrar usted mismo el inicio y funcionamiento del intérprete.*

También puede utilizar un archivo de configuración php.ini personalizado para, por ejemplo, tomar ventaja de los módulos adicionales. El archivo php.ini puede utilizarse particularmente para declarar la ubicación de las extensiones PHP.

*El archivo de inicialización php.ini se ubica en la carpeta **Resources** de la base. Si este archivo no está presente durante la primera llamada, 4D lo creará con las opciones de configuración apropiadas.*
El archivo php.ini del intérprete externo debe contener las siguientes entradas:

- ***auto_prepend_file** que ofrece la ruta de acceso completa al script utilitario 4D_Execute_PHP.php. Este script se encuentra en [aplicación 4D]Resources/php/Windows o /Mac. Sin esta entrada, sólo los scripts enteros pueden ejecutarse: la llamada de una rutina dentro de un script no funcionará.*
- ***display_errors** fija un "stderr" de manera que 4D pueda estar informado cuando ocurra un error durante la ejecución de código PHP. Ejemplo:*

```
; stderr - Mostrar los errores para STDERR (afecta únicamente los CGI/CLI)
Para dirigir los errores a STDERR para los CGI/CLI:
display_errors = "stderr"
```

Para mayor información sobre la configuración de archivos php.ini personalizados, por favor consulte los comentarios que se encuentran en el archivo php.ini suministrado por 4D.

PHP Execute

PHP Execute (rutaScript {; nomFuncion {; resultPHP {; param} {; param2 ; ... ; paramN}}}) -> Resultado

Parámetro	Tipo		Descripción
rutaScript	Texto	➡	Ruta de acceso al script PHP o "" para ejecutar una función PHP
nomFuncion	Texto	➡	Función PHP a ejecutar
resultPHP	Operador, Variable, Campo	⬅	Resultado de ejecución de la función PHP o * para no recibir el resultado
param	Texto, Booleano, Real, Entero largo, Fecha, Hora	➡	Parámetros de la función PHP
Resultado	Booleano	↩	True = ejecución correcta, False = error de ejecución

Descripción

*El comando **PHP Execute** permite ejecutar un script o una función PHP.*

Pase en el parámetro rutaScript la ruta de acceso del archivo de script PHP a ejecutar. Puede ser una ruta de acceso relativa si el archivo está ubicado junto a la estructura de la base o de una ruta absoluta. La ruta de acceso puede expresarse en sintaxis sistema o POSIX.
Si quiere ejecutar directamente una función PHP estándar, pase una cadena vacía ("") en rutaScript. El nombre de la función debe pasarse en el segundo parámetro.

Pase en el parámetro nomFuncion un nombre de función PHP si quiere ejecutar una función específica en el script rutaScript. Si pasa una cadena vacía u omite el parámetro nomFuncion, el script se ejecuta completamente.

***Nota:** PHP tiene en cuenta las mayúsculas y minúsculas de los caracteres en el nombre de la función. No utilice paréntesis, introduzca únicamente el nombre de la función.*

El parámetro resultPHP recibe el resultado de la ejecución de la función PHP. Puede pasar:

- *una variable, un array o un campo para recibir el resultado,*
- *el carácter * si la función no devuelve ningún resultado o si no quiere recuperar el resultado.*

El parámetro resultPHP puede ser de tipo texto, entero largo, real, booleano o fecha así como también (excepto para arrays) BLOB u hora. 4D efectuará la conversión de los datos y los ajustes necesarios siguiendo los principios descritos en la sección a continuación.

- *si pasa un nombre de función en el parámetro nomFuncion, resultPHP recibirá lo que el desarrollador PHP devuelve con el comando **return** del cuerpo de la función.*
- *Si utiliza el comando sin pasar un nombre de función en el parámetro nomFuncion, resultPHP recibirá lo que el desarrollador PHP devolvió con el comando **echo** (o un comando similar).*

*Si llama a una función PHP que espera argumentos, utilice los parámetros param1...N para pasar uno o varios valores. Los valores deben estar separados por punto y coma. Puede pasar valores de tipo alfa, texto, booleano, real, entero, entero largo, fecha u hora. Las imágenes y los BLOBs y objetos no se aceptan. Puede enviar un array; en este caso debe pasar un puntero en el array al comando **PHP Execute**, de lo contrario el índice actual del array se enviará como un entero (ver el ejemplo). El comando acepta todos los tipos de arrays excepto los arrays puntero, los arrays imagen y los arrays 2D.*

***Nota:** por razones técnicas, el tamaño de los parámetros pasados vía el protocolo FastCGI no debe pasar los 64 KB. Debe tener en cuenta esta limitación si utiliza parámetros de tipo Texto.*

*El comando devuelve True si la ejecución se ha efectuado correctamente del lado de 4D, en otras palabras, si el lanzamiento del entorno de ejecución, la apertura del script y el establecimiento de la comunicación con el intérprete PHP fueron exitosos. De lo contrario, se genera un error, que puede interceptar con el comando **ON ERR CALL** y analizar con **GET LAST ERROR STACK**. Además, el script mismo puede generar errores PHP. En este caso, debe utilizar el comando **PHP GET FULL RESPONSE** para analizar la fuente del error (ver ejemplo 4).*

***Nota:** PHP permite configurar la gestión de errores. Para mayor información, consulte por ejemplo la página: http://www.php.net/manual/en/errorfunc.configuration.php#ini.error-reporting.*

Conversión de los datos devueltos

La siguiente tabla especifica cómo 4D interpreta y convierte los datos devueltos en función del tipo del parámetro resultPHP.

Tipo del parámetro resultPHP	Proceso 4D	Ejemplo
BLOB	4D recupera los datos recibidos sin ninguna modificación(*).	
Texto	4D espera los datos codificados en UTF-8 (*). El desarrollador PHP puede necesitar utilizar el comando PHP **utf8_encode**.	Ejemplo de script PHP: `echo utf8_encode(miTexto)`
Fecha	4D espera una fecha enviada como una cadena en formato RFC 3339 (también llamado DATE_ATOM en PHP). Este formato es de tipo "AAAA-MM-DDTHH:MM:SS", por ejemplo: 2005-08-15T15:52:01+00:00. 4D ignora la parte hora y devuelve la fecha en UTC.	
Hora	4D espera una hora enviada en forma de cadena en formato RFC 3339 (ver el tipo Fecha). 4D ignora la parte fecha y devuelve el número de segundos transcurridos desde la media noche considerando la fecha en la zona horaria local.	Ejemplo de script PHP para enviar 2h30'45": `echo date( DATE_ATOM, mktime( 2,30,45))`
Entero o Real	4D interpreta el valor numérico expresado con números, signo + o - y/o el exponente con el prefijo 'e'. Todo carácter '.' o ',' se interpreta como un separador decimal.	Ejemplo de script PHP: `echo -1.4e-16;`
Booleano	4D devuelve True si recibe la cadena "true" desde PHP o si la evaluación numérica da un valor no nulo.	Ejemplo de script PHP: `echo (a==b);`
Array	4D considera que el array PHP fue devuelto en el formato JSON.	Ejemplo de script PHP para devolver un array de dos textos: `echo json_encode( array( "hola", "mundo"));`

(*) Por defecto, no se devuelven los encabezados HTTP:
*- Si utiliza **PHP Execute** al pasar una función en el parámetro nomFuncion, los encabezados HTTP nunca se devuelven en resultPHP. Sólo están disponibles por medio del comando **PHP GET FULL RESPONSE**.*
*- Si utiliza **PHP Execute** sin un nombre de función (el parámetro nomFuncion se omite o contiene una cadena vacía), puede devolver los encabezados HTTP fijando la opción PHP Raw result en True utilizando el comando **PHP SET OPTION**.*

***Nota:** si debe recuperar grandes volúmenes de datos utilizando PHP, es más eficiente pasar por el canal del buffer stdOut (comando **echo** o similar) que por el retorno de función. Para mayor información, consulte la descripción del comando **PHP GET FULL RESPONSE**.*

Using environment variables

*Puede utilizar el comando **SET ENVIRONMENT VARIABLE** para definir las variables de entorno utilizadas por el script. Atención: después de llamar **LAUNCH EXTERNAL PROCESS** o **PHP Execute**, el conjunto de las variables entorno se borra.*

Special functions

4D ofrece las siguientes funciones especiales:

- ***quit_4d_php**: permite salir del intérprete PHP y de todos sus procesos hijos. Si al menos uno de los procesos hijo está ejecutando un script, el intérprete no sale y el comando **PHP Execute** devuelve False.*
- ***relaunch_4d_php**: permite relanzar el intérprete PHP.*

*Note que el intérprete se relanza automáticamente cuando la primera petición se envía por **PHP Execute**.*

Ejemplo 1

Llamada del script "myPhpFile.php" sin función. Este es el contenido del script:

```
<?php
echo 'Versión PHP actual: ' . phpversion();
?>
```

El siguiente código 4D:

```
C_TEXT($result)
C_BOOLEAN($isOK)
$isOK:=PHP Execute("C:\php\myPhpFile.php";"";$result)
ALERT($Result)
```

... mostrará "Versión PHP actual: 5.3"

Ejemplo 2

Llamada de la función myPhpFunction en el script "myNewScript.php" con parámetros. Este es el contenido del script:

```
<?php
// . . . código PHP. . .
function myPhpFunction($p1, $p2) {
    return $p1 . ' '. $p2;
}
// . . . código PHP.. . .
?>
```

Llamada con función:

```
C_TEXT($result)
C_TEXT($param1)
C_TEXT($param2)
C_BOOLEAN($isOk)
$param1 :="¡Hola"
$param2 :="mundo 4D!"
$isOk:=PHP Execute("C:\MyFolder\myNewScript.php";"myPhpFunction";$result;$param1;$param2)
ALERT($result) // Muestra "¡Hola mundo 4D!"
```

Ejemplo 3

Salir del intérprete PHP:

```
$ifOk:=PHP Execute("";"quit_4d_php")
```

Ejemplo 4

Gestión de errores:

```
 // Instalación del método de gestión de errores
phpCommError:="" // Modificado por PHPErrorHandler
$T_saveErrorHandler :=Method called on error
ON ERR CALL("PHPErrorHandler")</p><p> // Ejecución del script
C_TEXT($T_result)
If(PHP Execute("C:\MyScripts\MiscInfos.php";"";$T_result))
 // Ningún error, $T_Result contiene el resultado
Else
 // Se ha detectado un error, administrado por PHPErrorHandler
   If(phpCommError="")
      ... // error PHP, utilice PHP GET FULL RESPONSE
   Else
      ALERT(phpCommError)
   End if
End if

 // Desinstalación del método
ON ERR CALL($T_saveErrorHandler)
```

*El método **PHP_errHandler** es el siguiente:*

```
phpCommError:=""
GET LAST ERROR STACK(arrCodes;arrComps;arrLabels)
For($i;1;Size of array(arrCodes))
   phpCommError:=phpCommError+String(arrCodes{$i})+" "+arrComps{$i}+" "+
   arrLabels{$i}+Char(Carriage return)
End for
```

Ejemplo 5

Creación dinámica por 4D de un script antes de su ejecución:

```
DOCUMENT TO BLOB("C:\Scripts\MyScript.php";$blobDoc)
If(OK=1)
```

```
    $strDoc:=BLOB to text($blobDoc;UTF8 text without length)

    $strPosition:=Position("function2Rename";$strDoc)

    $strDoc:=Insert string($strDoc;"_v2";Length("function2Rename")+$strPosition)

    TEXT TO BLOB($strDoc;$blobDoc;UTF8 text without length)
    BLOB TO DOCUMENT("C:\Scripts\MyScript.php";$blobDoc)
    If(OK#1)
        ALERT("Error on script creation")
    End if
End if
```

Se ejecuta el script:

```
$err:=PHP Execute("C:\Scripts\MyScript.php";"function2Rename_v2";*)
```

Ejemplo 6

Recuperación directa de un valor de tipo fecha y hora. Este es el contenido del script:

```
<?php
// . . . code php. . .
echo date(DATE_ATOM, mktime(1, 2, 3, 4, 5, 2009));
// . . . code php. . .
?>
```

Recepción de la fecha del lado 4D:

```
C_DATE($phpResult_date)
$result :=PHP Execute("C:\php_scripts\ReturnDate.php";"";$phpResult_date)
 //$phpResult_date is !05/04/2009 !

C_TIME($phpResult_time)
$result :=PHP Execute("C:\php_scripts\ReturnDate.php";"";$phpResult_time)
 //$phpResult_time is ?01 :02 :03 ?
```

Ejemplo 7

Distribución de datos en arrays:

```
ARRAY TEXT($arText ;0)
ARRAY LONGINT($arLong ;0)
$p1 :="," 
$p2 :="11,22,33,44,55"
$phpok :=PHP Execute("";"explode";$arText;$p1;$p2)
$phpok :=PHP Execute("";"explode";$arLong;$p1;$p2)

 // $arText contiene los valores Alfa "11", "22", "33", etc.
 // $arLong contiene los números, 11, 22, 33, etc.
```

Ejemplo 8

Inicialización de un array:

```
ARRAY TEXT($arText ;0)
$phpok :=PHP Execute("";"array_pad";$arText;->$arText;50;"indefinido")
 // Ejecute en PHP: $arrTest = array_pad($arrTest, 50, 'indefinido');
 // Llene el array $arText con 50 elementos "indefinido"
```

Ejemplo 9

Paso de parámetros vía un array:

```
ARRAY INTEGER($arInt;0)
$phpok :=PHP Execute("";"json_decode";$arInt;"[13,51,69,42,7]")
```

```
// Ejecute en PHP: $arInt = json_decode('[13,51,69,42,7]');
// Llene el array con los valores iniciales
```

Ejemplo 10

Este es un ejemplo de la utilización básica de la función trim, para eliminar espacios adicionales y/o caracteres invisibles de principio a fin de una cadena:

```
C_TEXT($T_String)
$T_String:="    Hello   "
C_BOOLEAN($B)
$B:=PHP Execute("";"trim";$T_String;$T_String)
```

Para obtener más información acerca de la función trim, por favor, consulte la documentación de PHP.

PHP GET FULL RESPONSE

PHP GET FULL RESPONSE (stdOut {; etiquetasErr ; valoresErr} {; camposEncHttp {; valoresEncHttp}})

Parámetro	Tipo		Descripción
stdOut	Variable texto, BLOB variable	⇐	Contenido del buffer stdOut
etiquetasErr	Array texto	⇐	Etiquetas de los errores
valoresErr	Array texto	⇐	Valores de los errores
camposEncHttp	Array texto	⇐	Nombres de los encabezados HTTP
valoresEncHttp	Array texto	⇐	Valores de los encabezados HTTP

Descripción

*El comando **PHP GET FULL RESPONSE** permite obtener información adicional sobre la respuesta devuelta por el intérprete PHP. Este comando es especialmente útil en el caso de que ocurra un error durante la ejecución del script.*

*El script PHP puede escribir datos en el buffer stdOut (eco, print, etc.) El comando devuelve directamente los datos en la variable stdOut y aplica los mismos principios de conversión descritos en el comando **PHP Execute**.*

Los arrays texto sincronizados etiquetasErr y valoresErrV se llenan cuando la ejecución de los scripts PHP provoca errores. Estos arrays, en particular, proporcionan información sobre el origen, el script y la línea de error. Estas dos arrays son inseparables: si se pasa etiquetasErr, se debe pasar también valoresErr.

Dado que los intercambios entre 4D y el intérprete PHP se efectúan a través de FastCGI, el intérprete PHP funciona como si fuera llamado por un servidor HTTP y por tanto, envía encabezados HTTP. Puede recuperar estos encabezados y sus valores en los arrays camposEncHttp y valoresEncHttp.

PHP GET OPTION

PHP GET OPTION (opcion ; valor)

Parámetro	Tipo		Descripción
opcion	Entero largo	➡	Opción a leer
valor	Texto, Booleano	⬅	Valor actual de la opción

Descripción

*El comando **PHP GET OPTION** permite conocer el valor actual de una opción relativa a la ejecución de scripts PHP.*

*Pase en el parámetro opcion una constante del tema "**PHP**" para designar la opción a leer. El comando devuelve el valor actual de la opción en el parámetro valor. Puede pasar una de las siguientes constantes:*

Constante	Tipo	Valor
PHP privileges	Entero largo	1
PHP raw result	Entero largo	2

***Nota:** sólo la cuenta de usuario se devuelve cuando utiliza la opción PHP Privileges con el comando **PHP GET OPTION** (no se devuelve la contraseña).*

Ejemplo

Queremos conocer la cuenta de usuario actual:

```
C_TEXT($userAccount)
PHP GET OPTION(PHP privileges;$userAccount)
ALERT($userAccount)
```

PHP SET OPTION

PHP SET OPTION (opción ; valor {; *})

Parámetro	Tipo		Descripción
opción	Entero largo	➡	Número de opción a definir
valor	Texto, Booleano	➡	Nuevo valor de la opción
*	Operador	➡	Si se pasa: la modificación no se aplica a la siguiente llamada

Descripción

*El comando **PHP SET OPTION** se utiliza para definir opciones específicas antes de llamar el comando **PHP Execute**. El alcance de este comando es el proceso actual.*

*Pase en el parámetro opcion una constante del tema "**PHP**" para designar la opción a modificar y en el parámetro valor, el nuevo valor de la opción. Esta es una descripción de las opciones:*

Constante	Tipo	Valor
PHP privileges	Entero largo	1
PHP raw result	Entero largo	2

*Por defecto, **PHP SET OPTION** define la opción para todas las llamadas a **PHP Execute** posteriores del proceso. Si lo quiere definir para la próxima llamada únicamente pase el parámetro estrella (*).*

Ejemplo

Ejecute el script "myAdminScript.php" con los derechos de acceso Admin:

```
PHP SET OPTION(PHP privileges;"admin:mypwd";*)
  `Como pasamos *, los privilegios admin se utilizarán una sola vez
C_TEXT($result)
C_BOOLEAN($isOK)
$isOK:=PHP Execute("myAdminScript.php";$result)
If($isOK)
    ALERT($result)
End if
```

Soporte de módulos PHP

Este anexo detalla la implementación de los módulos PHP en 4D. Se tratan los siguientes temas:

- *Lista de módulos estándar PHP ofrecidos por defecto con el intérprete PHP de 4D*
- *Lista de los módulos estándar PHP no retenidos por 4D*
- *Modificaciones del archivo php.ini*

*Nota: si desea instalar módulos adicionales, debe utilizar un intérprete FastCGI-php externo (ver **Utilizar otro intérprete PHP y otro archivo php.ini**).*

Módulos ofrecidos por defecto

La siguiente tabla detalla los módulos PHP ofrecidos por defecto con 4D.

Módulos genéricos

Nombre	Sitio web	Descripción

BCMath	http://php.net/bc	Calculadora binaria que maneja números de cualquier tamaño y precisión representados como cadenas.
		Ejemplo:
		`C_LONGINT($valor;$result)`
		`$valor:=4`
		`$ok:=PHP Execute("";"bcpow";$result;$valor;3)`

Calendar	http://php.net/calendar	Conjunto de funciones que simplifican la conversión entre los diferentes formatos de calendarios. Basado en Julian Day Count.
		Ejemplo:
		`C_LONGINT($NumeroDeDias)`
		`$ok:=PHP Execute("";"cal_days_in_month";$NumeroDeDias;1;2;2010)`

Ctype	http://php.net/ctype	Funciones que verifican si un carácter o una cadena pertenecen a una cierta clase de caracteres, dependiendo de la configuración local actual

[tab/]	Ejemplo:	`// Verificar que todos los caracteres de la cadena sean signos de puntuación`
		`C_TEXT($myString)`
		`$myString:=",.;/"`
		`$ok:=PHP Execute("";"ctype_punct";$isPunct;$myString)`

Date and Time	http://php.net/datetime	Recuperación de la fecha y hora desde el servidor donde el script PHP se ejecuta
		Ejemplo:
		`//Cálculo de la hora del amanecer en Lisboa, Portugal`
		`//Latitud: 38.4 North`
		`//Longitud: 9 West`
		`//Zenith ~= 90`
		`//Time-lag: +1 GMT`
		`C_TIME($SunriseTime)`
		`$ok:=PHP Execute("";"date_sunrise";$SunriseTime;0;1;38,41;-9;90;1)`

DOM (Document Object Model)	http://php.net/dom	Uso de documentos XML vía el API DOM API de PHP 5
Exif	http://php.net/exif	Trabaja con los metadatos de las imágenes.
Fileinfo(*)	http://php.net/fileinfo	Detección del tipo de contenido y de la codificación de un archivo.

Filter	http://php.net/filter	Validar y filtrar los datos de una fuente no segura, como las entradas de los usuarios.
		Ejemplo:
		`C_LONGINT($filterId)`
		`C_TEXT($result)`
		`$ok:=PHP Execute("";"filter_id";$filterId;"validate_email")`
		`$ok:=PHP Execute("";"filter_var";$result;"hop@123.com";$filterId)`

FTP (File Transfert Protocol)	http://php.net/ftp	Acceso detallado a un servidor FTP

Hash	http://php.net/hash	Motor de resumen de mensajes (Message Digest). Permite el procesamiento directo o indirecto del mensaje de longitud arbitrario utilizando una variedad de algoritmos
		Ejemplo:
		`C_TEXT($md5Result)`
		`$ok:=PHP Execute("";"md5";$md5Result;"esta es mi cadena a la que se le va a hacer hash")`

GD (Graphics Draw) Library	http://php.net/gd	Manipulación de imágenes
Iconv	http://php.net/iconv	Conversión de archivos entre diferentes conjuntos de caracteres
JSON (JavaScript Object Notation)	http://php.net/json	Implementación del formato de intercambio de datos JSON
LDAP	http://php.net/ldap	LDAP es un protocolo de acceso a los "servidores de carpetas" que almacenan la

		información en forma de diagrama de árbol
LibXML	*http://php.net/libxml*	*Librería de funciones y constantes XML*
LibXSLT	*http://php.net/xsl*	*Librería de funciones de transformación XSL*
Multibyte String	*http://php.net/mbstring*	*Conjunto de funciones de manipulación de cadenas que permite trabajar con las codificaciones multi-bytes o convertir los conjuntos de caracteres.*
OpenSSL	*http://php.net/openssl*	*Uso de las funciones de OpenSSL para generar y verificar las firmas, cifrar y descifrar los datos.*
PCRE (Perl Compatible Regular Expressions)	*http://php.net/pcre*	*Conjunto de funciones que implementan las expresiones racionales utilizando la misma sintaxis y semántica que Perl 5*

Ejemplo:

```
// Este ejemplo remueve los espacios innecesarios de una cadena
C_TEXT($myString)
$myString:="foo        o        bar"
PHP Execute("";"preg_replace";$myString;"/\\s\\s+/";" ";$myString)
ALERT($myString)
//La cadena contiene ahora 'foo o bar' sin espacios duplicados
```

PDO (PHP Data Objects)	*http://php.net/pdo*	*Interfaz de acceso a una base de datos. Necesita un driver PDO específico a la base de datos.*
PDO_SQLITE	*http://php.net/pdo_sqlite*	*Driver que implementa la interfaz de PHP Data Objects (PDO) para autorizar el acceso de PHP a las bases de datos SQLite 3.*
Reflection	*http://php.net/reflection*	*API de reflexión completa que permite reverse-engineering en las clases, las interfaces, las funciones, los métodos, como también las extensiones*
Phar (PHP Archive)	*http://php.net/phar*	*Permite incluir una aplicación PHP completa en un archivo único llamado "phar" (PHP Archive) para facilitar su instalación y su configuración*
Session	*http://php.net/session*	*Soporte de sesiones PHP*

Ejemplo:

*Las sesiones se utilizan en las aplicaciones web para conservar el contexto entre cada consulta. Cuando usted llama **PHP Execute** en 4D, el script PHP puede iniciar una sesión y almacenar todo lo que es útil para conservar como contexto en el array asociado $_SESSION. Si un script PHP utiliza las sesiones, debe obtener el ID de sesión devuelto por PHP utilizando el comando **PHP GET FULL RESPONSE** y definir antes de cada llamada a **PHP Execute** utilizando el comando **SET ENVIRONMENT VARIABLE**.*

```
// Método "PHP Ejecutar con contexto"
If(<>PHP_Session#"")
   SET ENVIRONMENT VARIABLE("HTTP_COOKIE";<>PHP_Session)
End if
If(PHP Execute($1))
   PHP GET FULL RESPONSE($0;$errorInfos;$errorValues;$headerFields;$headerValues)
   $idx:=Find in array($headerFields;"Set-Cookie")
   If($idx>0)
      <>PHP_Session:=$headerValues{$idx}
   End if
End if
```

SimpleXML	*http://php.net/simpleXML*	*Herramientas muy simples y fáciles de utilizar para convertir de XML a un objeto que puede ser procesado con sus propiedades y los iteradores de arrays*
Sockets	*http://php.net/sockets*	*Implementación de una interfaz de bajo nivel con las funciones de comunicación por socket basados en los sockets BSD y ofrece la posibilidad de funcionar también como servidor socket y cliente.*
SPL (Standard PHP Library)	*http://php.net/spl*	*Colección de interfaces y de clases utilitarias creadas para resolver problemas estándar.*
SQLite	*http://php.net/sqlite*	*Extensión para el motor de base de datos SQLite. Este motor puede estar embebido.*
SQLite3	*http://php.net/sqlite3*	*Soporte para las bases de datos SQLite versión 3*
Tokenizer	*http://php.net/tokenizer*	*Funciones que permiten escribir sus propias herramientas PHP de análisis o de modificaciones sin tener que tratar con la especificación del lenguaje al nivel lexical*
XML (eXtensible Markup Language)	*http://php.net/xml*	*Análisis de los documentos XML*
XMLreader	*http://php.net/xmlreader*	*Analizador XML Pull*
XMLwriter	*http://php.net/xmlwriter*	*Generación del flujo y de los archivos al formato XML*
Zlib	*http://php.net/zlib*	*Lectura y escritura de archivos comprimidos gzip (.gz)*

Ejemplo:

```
WEB GET HTTP HEADER($names;$values)
```

```
$pos:=Find in array($names;"Accept-Encoding")
If($pos>0)
    Case of
        :(Position($values{$pos};"gzip")>0)
            WEB SET HTTP HEADER("Content-Encoding: gzip")
            PHP Execute("";"gzencode";$html;$html)
        :(Position($values{$pos};"deflate")>0)
            WEB SET HTTP HEADER("Content-Encoding: deflate")
            PHP Execute("";"gzdeflate";$html;$html)
    End case
End if
WEB SEND TEXT($html)
```

Zip	*http://php.net/zip*	*Lectura y escritura de los archivos comprimidos ZIP y los archivos en él*

(*) En la versión actual de 4D, estos módulos no están disponibles bajo Windows

Módulos disponibles bajo Windows únicamente

Por razones estructurales, los siguientes módulos PHP sólo están disponibles en la plataforma Windows.

Nombre	**Sitio web**	**Descripción**
COM & .NET	*http://php.net/com*	*COM (Component Object Model) es uno de los métodos más utilizados por aplicaciones y componentes para la comunicación en plataformas Windows. Adicionalmente, 4D soporta la ejemplificación y creación de ensambles .Net vía la capa COM.*
ODBC (Open DataBase Connectivity)	*http://php.net/odbc*	*Además del soporte del ODBC estándar, el ODBC unificado de PHP le da acceso a varias bases de datos que han tomado prestada la semántica de los API ODBC para implementar sus propios API.*
WDDX (Web Distributed Data eXchange)	*http://php.net/wddx*	*Facilita los intercambios de datos entre aplicaciones web vía la web, sin importar la plataforma*

Módulos desactivados

Los siguientes módulos PHP no se han implementado en 4D. La columna derecha explica la razón por la cual no se implementó:

Nombre	**Sitio web**	**Causa - Solución alternativa**
Mimetype	*http://php.net/mime-magic*	*Obsoleto - Utilizar Fileinfo*
POSIX (Portable Operating System Interface)	*http://php.net/posix*	*Obsoleto*
Regular Expression (POSIX Extended)	*http://php.net/regex*	*Obsoleto - Utilizar PCRE*
Crack	*http://php.net/crack*	*Licencia restrictiva*
ffmpeg	*http://ffmpeg-php.sourceforge.net/*	*Licencia restrictiva - Uso ffmpeg en línea de comando con LAUNCH EXTERNAL PROCESS*
Image Magick	*http://php.net/manual/book.imagick.php*	*Licencia restrictiva - Uso GD 2*
IMAP (Internet Message Access Protocol)	*http://php.net/imap*	*Licencia restrictiva - Uso del plug-in integrado 4D Internet Commands*
PDF (Portable Document Format)	*http://php.net/pdf*	*Licencia restrictiva - Utilizar Haru PDF*
Mysqlnd (MySQL Native Driver)	*http://dev.mysql.com/downloads/connector/php-mysqlnd/*	*No pertinente en el entorno 4D*
Phar (PHP Archive)	*http://php.net/phar*	*No pertinente en el entorno 4D*

Personalizar el archivo php.ini

*El archivo "php.ini" a modificar (ver más adelante) puede estar ubicado en la carpeta **Resources\php** de la aplicación 4D (archivo por defecto) o en la carpeta **Resources** de la base (archivo personalizado). Para más información sobre este tema, consulte **Ejecutar scripts PHP en 4D**.*

***Advertencia**: la modificación del archivo "php.ini" debe hacerse con precaución y requiere un buen conocimiento de PHP. Para más información acerca de la configuración de los archivos php.ini personalizados, puede consultar los comentarios que se encuentran en el archivo php.ini ofrecido por 4D.*

***Nota**: si la duración del procesamiento de PHP es relativamente larga (más allá de 30 segundos), por defecto se devuelve un error de tipo 'timeout' en 4D y el procesamiento fallará. En este caso, puede configurar el timeout por defecto con el fin de dedicar más tiempo a la ejecución PHP. Hay dos maneras de hacer esto:*

- *definiendo la variable **max_execution_time** en el archivo "php.ini" (pase un valor en segundos). Atención: esta configuración afecta a todas los scripts*
- *llamando el comando **set_time_limit (nbSec)** en el script de ejecución PHP que está realizando el proceso largo. Pase el tiempo máximo asignado a la ejecución del script PHP en nbSec. Se recomienda utilizar esta configuración, ya que sólo afecta a este script. Por lo general, por razones de seguridad, es preferible conservar un valor de timeout bajo para los scripts PHP.*

Portapapeles

Gestión de portapapeles
APPEND DATA TO PASTEBOARD
CLEAR PASTEBOARD
Get file from pasteboard
GET PASTEBOARD DATA
GET PASTEBOARD DATA TYPE
GET PICTURE FROM PASTEBOARD
Get text from pasteboard
Pasteboard data size
SET FILE TO PASTEBOARD
SET PICTURE TO PASTEBOARD
SET TEXT TO PASTEBOARD

Gestión de portapapeles

Los comandos del tema "Portapapeles" pueden gestionar las acciones copiar/pegar (gestión de Portapapeles), como también las acciones arrastrar y soltar entre aplicaciones.

4D utiliza dos portapapeles: uno par los datos copiados (o cortados), que es el portapapeles que ya estaba presente en las versiones anteriores, y el otro para los datos que están siendo arrastrados y soltados.

Estos dos portapapeles son administrados utilizando los mismos comandos. Usted accede a uno o a otro dependiendo del contexto:

- *El portapapeles de arrastrar y soltar es accesible únicamente en el caso de los eventos de formulario **On Begin Drag Over**, **On Drag over** o **On Drop** y en el **Método de base On Drop**. Fuera de estos contextos, el portapapeles arrastrar y soltar no está disponible.*
- *Se puede acceder al portapapeles copiar/pegar en todos los demás casos. A diferencia del portapapeles arrastrar y soltar, conserva durante toda la sesión los datos que fueron colocados en él, siempre no sean borrados o reutilizados.*

Tipos de datos

Durante las acciones arrastrar y soltar, diferentes tipos de datos pueden colocarse y leerse en el portapapeles. Puede acceder a un tipo de datos de varias formas:

- *Vía su firma 4D: la firma 4D es una cadena de caracteres que indica un tipo de datos referenciado por 4D. El uso de firmas 4D facilita el desarrollo de aplicaciones multiplataforma ya que estas firmas son idénticas en Mac OS y Windows. Más adelante encontrará la lista de firmas 4D.*
- *Vía un UTI (Uniform Tipo Identifier, Mac OS únicamente): la norma UTI, definida por Apple, asocia una cadena de caracteres con cada tipo de objeto nativo. Por ejemplo, las imágenes GIF tienen el tipo UTI "com.apple.gif". Los tipos UTI son publicados en la documentación de Apple como también por lo editores relacionados.*
- *Vía su número or su nombre de formato (Windows únicamente): bajo Windows, cada tipo de dato nativo está referenciado por su número ("3", "12", etc.) y un nombre ("Rich Text Edit"). Por defecto, Microsoft especifica varios tipos nativos llamados formatos de datos estándar. Adicionalmente, los editores de terceras partes pueden "guardar" nombres de formatos en el sistema, que les atribuye un número en retorno. Para mayor información sobre éste y otros tipos nativos, por favor consulte la documentación Microsoft developer (en particular http://msdn2.microsoft.com/en-us/library/ms649013.aspx).*

***Nota:** en los comandos de 4D, los números de formatos Windows son manejados como texto.*

*Todos los comandos del tema "Portapapeles" pueden trabajar con cada uno de estos tipos de datos. Puede conocer los tipos de datos presentes en el portapapeles en cada uno de estos formatos utilizando **GET PASTEBOARD DATA TYPE**.*

***Nota:** los tipos de 4 caracteres (TEXT, PICT o tipos personalizados) se conservan por compatibilidad con las versiones anteriores de 4D.*

Firmas 4D

Esta es la lista de firmas 4D estándar así como su descripción:

Firma	Descripción
"com.4d.private.text.native"	Texto en conjunto de caracteres nativo
"com.4d.private.text.utf16"	Texto en conjunto de caracteres Unicode
"com.4d.private.text.rtf"	Texto enriquecido
"com.4d.private.picture.pict"	Imagen formato PICT
"com.4d.private.picture.pgn"	Imagen formato PNG
"com.4d.private.picture.gif"	Imagen formato GIF
"com.4d.private.picture.jfif"	Imagen formato JPEG
"com.4d.private.picture.emf"	Imagen formato EMF
"com.4d.private.picture.bitmap"	Imagen formato BITMAP
"com.4d.private.picture.tiff"	Imagen formato TIFF
"com.4d.private.picture.pdf"	Documento PDF
"com.4d.private.file.url"	Ruta de acceso al archivo

APPEND DATA TO PASTEBOARD

APPEND DATA TO PASTEBOARD (tipoDatos ; datos)

Parámetro	Tipo		Descripción
tipoDatos	Cadena	➡	Tipo de datos (4 caracteres)
datos	BLOB	➡	Datos a añadir al portapapeles

Descripción

*El comando **APPEND DATA TO PASTEBOARD** añade en el portapapeles los datos del tipo especificado en tipoDatos en el BLOB datos.*

***Nota:** en el caso de operaciones copiar/pegar, el contenedor de datos corresponde al Portapapeles.*

Pase en tipoDatos un valor definiendo el tipo de datos a añadir. Puede pasar una firma 4D, un tipo UTI (Mac OS), un nombre/número de formato (Windows), o un tipo de 4 caracteres (compatibilidad). Para mayor información sobre estos tipos, consulte la sección .

***Nota para los usuarios Windows**: cuando el comando se utiliza con datos de tipo texto (tipoDatos dataType rs "TEXT", com.4d.private.text.native o com.4d.private.text.utf16), la cadena contenida en el parámetro BLOB datos debe terminar con el carácter NULL en Windows.*

Si los datos del BLOB se añaden correctamente al portapapeles, la variable OK toma el valor 1. De lo contrario la variable OK toma el valor 0 y se puede generar un error.

*Generalmente, se utiliza el comando **APPEND DATA TO PASTEBOARD** para agregar múltiples instancias de los mismos datos al portapapeles o para añadir datos que no son de tipo TEXT o PICT. Para añadir nuevos datos al portapapeles, primero debe limpiar el portapapeles utilizando el comando **CLEAR PASTEBOARD**.*

Si quiere limpiar y añadir:

- *texto al portapapeles, utilice el comando **SET TEXT TO PASTEBOARD**,*
- *una imagen al portapapeles, utilice el comando **SET PICTURE TO PASTEBOARD**.*

*Sin embargo, note que si un BLOB contiene texto o una imagen, usted puede utilizar el comando **APPEND DATA TO PASTEBOARD** para añadir un texto o una imagen al portapapeles.*

Ejemplo

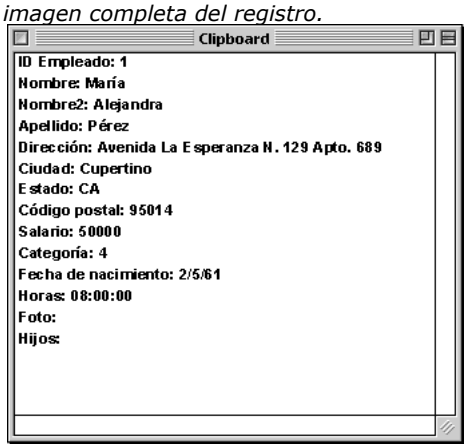
*Utilizando los comandos del tema portapapeles y de los BLOBs, puede construir esquemas sofisticados de Cortar/Copiar/Pegar para administrar datos estructurados en lugar de una sola pieza de información. En el siguiente ejemplo, los dos métodos de proyecto **SET RECORD TO CLIPBOARD** y **GET RECORD FROM CLIPBOARD** le permiten tratar un registro de una información a copiar en o desde el portapapeles.*

```
  ` Método de proyecto ENVIAR REGISTRO AL PORTAPAPELES
  ` ENVIAR REGISTRO AL PORTAPAPELES ( Numérico )
  ` ENVIAR REGISTRO AL PORTAPAPELES ( Número de tabla )

C_LONGINT($1;$vlCampo;$vlTipoCampo)
C_POINTER($vpTabla;$vpCampo)
C_STRING(255;$vsDocNombre)
C_TEXT($vtRegistroDatos;$vtCampoDatos)
C_BLOB($vxRegistroDatos)

  ` Limpiar el portapapeles (estará vacío si no hay un registro actual)
CLEAR PASTEBOARD
  ` Obtener un puntero a la tabla cuyo número se pasa como parámetro
$vpTabla:=Table($1)
  ` Si hay un registro actual para esa tabla
If((Record number($vpTabla->)>=0)|(Is new record($vpTabla->)))
  ` Inicializar la variable texto que contendrá la imagen de texto del registro
   $vtRegistroDatos:=""
  ` Para cada campo del registro:
   For($vlCampo;1;Count fields($1))
  ` Obtener el tipo de campo
      GET FIELD PROPERTIES($1;$vlCamp;$vlCampoTipo)
  ` Obtener un puntero hacia el campo
      $vpCampo:=Field($1;$vlCampo)
  ` Dependiendo del tipo de campo, copiar (o no) sus datos de manera apropiada
      Case of
         :(($vlCampoTipo=Is alpha field)|($vlCampoTipo=Is text))
            $vtCampoDatos:=$vpCampo->
         :(($vlCampoTipo=Is real)|($vlCampoTipo=Is integer)|($vlCampoTipo=Is longint)|($vlCampoTipo=Is date)|($vlCampoTipo=Is time))
            $vtCampoDatos:=String($vpCampo->)
         :($vlCampoTipo=Is Boolean)
```

```
            $vtCampoDatos:=String(Num($vpCampo->);"Sí;;No")
        Else
    ` Pasar e ignorar los otros tipos de campos
            $vtCampoDatos:=""
        End case
    ` Acumular los datos del campo en una variable de texto que almacena la imagen de texto del registro
        $vtRegistroDatos:=$vtRegistroDatos+Field name($1;$vlCampo)+":"+Char(9)+$vtCampoDatos+CR
    ` Nota: El método CR devuelve Char(13) en Macintosh y Char(13)+Char(10) en Windows
    End for
    ` Colocar la imagen de texto del registro en el portapapeles
    SET TEXT TO PASTEBOARD($vtRegistroDatos)
    ` Nombre del archivo scrap en la carpeta Temporales
    $vsDocNombre:=Temporary folder+"Scrap"+String(1+(Random%99))
    ` Borrar el archivo scrap si existe (Se debe hacer una prueba de error aquí)
    DELETE DOCUMENT($vsDocNombre)
    ` Crear archivo scrap
    SET CHANNEL(10;$vsDocNombre)
    `Enviar el registro completo al archivo scrap
    SEND RECORD($vpTabla->)
    ` Cerrar el archivo scrap
    SET CHANNEL(11)
    ` Cargar el archivo scrap en un BLOB
    DOCUMENT TO BLOB($vsDocNombre;$vxRegistroDatos)
    ` No necesitamos más el archivo scrap
    DELETE DOCUMENT($vsDocNombre)
    ` Añadir la imagen completa del registro al portapapeles
    ` Nota: utilizamos arbitrariamente el tipo de datos "4Drc"
    APPEND DATA TO PASTEBOARD("4Drc";$vxRegistroDatos)
    ` En este punto, el portapapeles contiene:
    ` (1) Una imagen de texto del registro (como se muestra en las copias de pantalla a continuación)
    ` (2) Una imagen completa del registro (incluyendo imágenes, subarchivos y los campos de tipo BLOB)
End if
```

Al introducir el siguiente registro:

Si aplica el métodoENVIAR REGISTRO AL portapapeles a la tabla [Empleados], el portapapeles contendrá el texto del registro y la imagen completa del registro.



*Puede pegar esta imagen del registro en otro registro, utilizando el método **GET RECORD FROM CLIPBOARD**, como se muestra a continuación:*

```
  ` Método OBTENER REGISTRO DESDE PORTAPAPELES
  ` OBTENER REGISTRO DESDE PORTAPAPELES ( Número )
  ` OBTENER REGISTRO DESDE PORTAPAPELES ( Número de tabla )
C_LONGINT($1;$vlCampo;$vlCampoTipo;$vlPosCR;$vlPosColon)
C_POINTER($vpTabla;$vpCampo)
C_STRING(255;$vsDocNombre)
C_BLOB($vxPortapapelesDatos)
C_TEXT($vtPortapapelesDatos;$vtCampoDatos)

  ` Obtener un puntero hacia la tabla cuyo número se pasa como parámetro
$vpTabla:=Table($1)
  ` Si hay un registro actual
If((Record number($vpTabla->)>=0)|(Is new record($vpTabla->)))
   Case of
  ` ¿El portapapeles contiene una imagen completa del registro?
      :(Pasteboard data size("4Drc")>0)
  ` Si es así, extraiga el contenido del portapapeles
         GET PASTEBOARD DATA("4Drc";$vxPortapapelesDatos)
  ` Nombre para el archivo scrap en la carpeta temporales
         $vsDocNombre:=Temporary folder+"Scrap"+String(1+(Random%99))
  ` Borrar el archivo scrap si existe (Se debe hacer una prueba de error aquí)
         DELETE DOCUMENT($vsDocNombre)
  ` Guardar el BLOB en el archivo scrap
         BLOB TO DOCUMENT($vsDocNombre;$vxPortapapelesDatos)
  ` Abrir el archivo scrap
         SET CHANNEL(10;$vsDocNombre)
  ` Recibir el registro completo del archivo scrap
         RECEIVE RECORD($vpTable->)
  ` Cerrar el archivo scrap
         SET CHANNEL(11)
  ` No necesitamos más el archivo scrap
         DELETE DOCUMENT($vsDocNombre)
  ` ¿El portapapeles contiene TEXT?
      :(Pasteboard data size("TEXT")>0)
  ` Extraer el texto del portapapeles
         $vtPortapapelesDatos:=Get text from pasteboard
  ` Inicializar el número de campos a incrementar
         $vlCampo:=0
         Repeat
  ` Buscar la línea de campo siguiente en el texto
            $vlPosCR:=Position(CR;$vtPortapapelesDatos)
            If($vlPosCR>0)
  ` Extraer la línea de campo
               $vtCampoDatos:=Substring($vtPortapapelesDatos;1;$vlPosCR-1)
  ` Si hay dos puntos ":"
               $vlPosColon:=Position(":";$vtCampoDatos)
               If($vlPosColon>0)
  ` Tomar sólo los datos de campo (eliminar el nombre de campo)
                  $vtCampoDatos:=Substring($vtCampoDatos;$vlPosColon+2)
               End if
  ` Incrementar el número de campo
               $vlCampo:=$vlCampo+1
  ` El portapapeles puede contener más información de la que necesitamos...
               If($vlCampo<=Count fields($vpTabla))
  ` Obtener el tipo de campo
                  GET FIELD PROPERTIES($1;$vlCampo;$vlCampoTipo)
  ` Obtener un puntero al campo
                  $vpCampo:=Field($1;$vlCampo)
  ` Dependiendo del tipo de campo, copiar (o no) el texo de una manera apropiada
                  Case of
                     :(($vlCampoTipo=Is alpha field)|($vlCampoTipo=Is text))
                        $vpCampo->:=$vtCampoDatos
                     :(($vlCampoTipo=Is real)|($vlCampoTipo=Is integer)|($vlCampoTipo=Is longint))
                        $vpCampo->:=Num($vtCampoDatos)
                     :($vlCampoTipo=Is date)
                        $vpCampo->:=Date($vtCampoDatos)
                     :($vlCampoTipo=Is time)
                        $vpCampo->:=Time($vtCampoDatos)
```

```
                    :($vlCampoTipo=Is Boolean)
                        $vpCampo->:=($vtCampoDatos="Sí")
                    Else
    ` Pasar e ignorar los otros tipos de datos de campos
                End case
            Else
    ` Todos los campos han sido asignados, salir del bucle
                $vtPortapapelesDatos:=""
            End if
    ` Eliminar el texto que acaba de ser extraído
            $vtPortapapelesDatos:=Substring($vtPortapapelesDatos;$vlPosCR+Length(CR))
        Else
    ` No se encontró un delimitador, salir del bucle
            $vtPortapapelesDatos:=""
        End if
    ` Repetir mientras tengamos datos
    Until(Length($vtPortapapelesDatos)=0)
Else
    ALERT("El portapapeles no contiene datos que puedan pegarse como un registro.")
End case
End if
```

Variables y conjuntos del sistema

Si los datos en el BLOB se añaden correctamente al portapapeles, la variable sistema OK toma el valor 1; de lo contrario OK toma el valor 0 y se podría generar un error.

CLEAR PASTEBOARD

CLEAR PASTEBOARD
Este comando no requiere parámetros

Descripción

*El comando **CLEAR PASTEBOARD** borra el contenido del portapapeles. Si el portapapeles contiene múltiples instancias de los mismos datos, todas las instancias se borran. Después de llamar a **CLEAR PASTEBOARD**, el portapapeles queda vacío.*

*Debe llamar **CLEAR PASTEBOARD** antes de añadir nuevos datos al portapapeles utilizando el comando **APPEND DATA TO PASTEBOARD**, porque este último comando no limpia el portapapeles antes de añadir nuevos datos.*

*Si llama a **CLEAR PASTEBOARD** una vez y luego llama a **APPEND DATA TO PASTEBOARD** muchas veces puede cortar o copiar los mismos datos bajo diferentes formatos.*

*Por el contrario, los comandos **SET TEXT TO PASTEBOARD** y **SET PICTURE TO PASTEBOARD** limpian automáticamente el portapapeles antes de poner datos TEXT o PICT en él.*

Ejemplo 1

El siguiente código borra y luego añade datos al portapapeles:

```
CLEAR PASTEBOARD  `Borra el contenido del portapapeles
APPEND DATA TO PASTEBOARD('XWKZ';$vxAlgunDato)  ` Añade datos de tipo 'XWKZ'
APPEND DATA TO PASTEBOARD('SYLK';$vxSylkDatos)  ` Añade datos de tipo SYLK
```

Ejemplo 2

*Ver el ejemplo del comando **APPEND DATA TO PASTEBOARD**.*

Get file from pasteboard

Get file from pasteboard (indiceN) -> Resultado

Parámetro	Tipo		Descripción
indiceN	Entero largo	➡	N archivo incluido en la acción arrastrar
Resultado	Cadena	⮌	Ruta de acceso al archivo extraído del portapapeles

Descripción

*El comando **Get file from pasteboard** devuelve la ruta de acceso absoluto de un archivo incluido en una operación de arrastrar y soltar. Varios archivos pueden ser seleccionados y movidos simultáneamente. El parámetro indiceN se utiliza para designar un archivo entre un conjunto de archivos seleccionados.*

Si no hay archivo N en el portapapeles, el comando devuelve una cadena vacía.

Ejemplo

El siguiente ejemplo puede utilizarse para recuperar en un array todas las rutas de acceso a los archivos incluídos en la operación arrastrar y soltar:

```
ARRAY TEXT($arrayArchivos;0)
C_TEXT($vtarchivo)
C_INTEGER($n)
$n:=1
Repeat
   $vtarchivo:=Get file from pasteboard($n)
   If($vtarchivo#"")
      APPEND TO ARRAY($arrayArchivos;$vtarchivo)
      $n:=$n+1
   End if
Until($vtarchivo="")
```

GET PASTEBOARD DATA

GET PASTEBOARD DATA (tipoDatos ; datos)

Parámetro	Tipo		Descripción
tipoDatos	Cadena	➡	Tipo de datos a extraer del contenedor
datos	BLOB	⬅	Datos extraídos del portapapeles

Descripción

*El comando **GET PASTEBOARD DATA** devuelve en el campo o variable de tipo BLOB datos que se encuentran en el portapapeles y cuyo tipo se pasa en tipoDatos. (Si el portapapeles contiene texto copiado en 4D, entonces el conjunto de caracteres del BLOB será probablemente UTF-16.)*

***Nota:** en el contexto de las operaciones de copiar/pegar, el contenedor corresponde al portapapeles.*

*Pase tipoDatos, pase un valor que defina el tipo de datos a extraer. Puede pasar una firma 4D, un tipo UTI (Mac OS), un nombre/número de formato (Windows), o un tipo de 4 caracteres (compatibilidad). Para mayor información sobre estos tipos, consulte la sección **Gestión de portapapeles**.*

***Nota:** no puede leer datos de tipo archivo con este comando; para hacer esto, debe utilizar el comando **Get file from pasteboard**.*

Ejemplo

Los métodos de objeto siguientes son los de dos botones que copian y pegan datos en el array asOpciones (menú pop-up, lista desplegable,...) ubicado en un formulario:

```
  ` Método de objeto bCopiar_asOpciones
If(Size of array(asOpciones)>0)  ` ¿Hay algo para copiar?
   VARIABLE TO BLOB(asOpciones;$vxClipData)  ` Acumular los elementos del array en un BLOB
   CLEAR PASTEBOARD  ` Vaciar el portapapeles
   APPEND DATA TO PASTEBOARD("artx";asOpciones)  ` Note que el tipo de datos se elige arbitrariamente
End if

  ` Método de objeto bPegar_asOpciones
If(Pasteboard data size("artx")>0)  ` Hay datos de tipo "artx" en el portapapeles?
   GET PASTEBOARD DATA("artx";$vxClipData)  ` Extraer los datos del portapapeles
   BLOB TO VARIABLE($vxClipData;asOpciones)  ` Llenar el array con los datos del BLOB
   asOpciones:=0  ` Reinicializar el elemento seleccionado del array
End if
```

Variables y conjuntos del sistema

Si los datos se extraen correctamente, la variable OK toma el valor 1; de lo contrario OK toma el valor 0 y se genera un error.

GET PASTEBOARD DATA TYPE

GET PASTEBOARD DATA TYPE (firmas4D ; tiposNativos {; nombresFormatos})

Parámetro	Tipo		Descripción
firmas4D	Array texto	⇐	Firmas 4D de tipos de datos
tiposNativos	Array texto	⇐	Tipos de datos nativos
nombresFormatos	Array texto	⇐	Nombres de los formatos (Windows únicamente), cadenas vacías bajo Mac OS

Descripción

*El comando **GET PASTEBOARD DATA TYPE** permite obtener la lista de los tipos de datos presentes en el portapapeles. Este comando generalmente debe ser utilizado en el contexto de una operación arrastrar y soltar, en los eventos de formulario **On Drop** o **On Drag Over** del objeto de destino. Más particularmente, permite verificar la presencia de un tipo de datos específico en el portapapeles.*

Este comando devuelve los tipos de datos en diferentes formas vía dos (o tres) arrays:

- *El array firmas4D contiene los tipos de datos expresados utilizando la firma 4D interna (por ejemplo, "com.4d.picture.gif"). Si 4D no reconoce un tipo de datos encontrado, una cadena vacía ("") se devuelve en el array.*
- *El array tiposNativos contiene los tipos de datos expresados utilizando su tipo nativo. El formato de los tipos nativos difiere entre Mac OS y Windows:*

- Bajo Mac OS, los tipos nativos son expresados como UTIs (Uniform Tipo Identifier).

- Bajo Windows, los tipos nativos son expresados como números, cada número está asociado a un nombre de formato. El array tiposNativos contiene estos números en forma de cadenas ("3", "12", etc.). Si quiere utilizar más etiquetas explícitas, se recomienda utilizar el array opcional nombresFormatos, que contiene el nombre del formato de los tipos nativos bajo Windows.

El array tiposNativos permite soportar todo tipo de datos presentes en el portapapeles, incluyendo los datos cuyo tipo no está referenciado por 4D.

- *Bajo Windows, también puede pasar el array nombresFormatos, que recibe los nombres de los tipos de datos encontrados en el portapapeles. Los valores devueltos en este array pueden ser utilizados por ejemplo para construir un menú desplegable de selección de formato. Bajo Mac OS, el array nomsFormats devuelve las cadenas vacías.*

*Para mayor información sobre los tipos de datos soportados, consulte la sección **Gestión de portapapeles**.*

GET PICTURE FROM PASTEBOARD

GET PICTURE FROM PASTEBOARD (imagen)

Parámetro	Tipo		Descripción
imagen	Imagen	⇐	Imagen extraída del Portapapeles

Descripción

GET PICTURE FROM PASTEBOARD devuelve la imagen presente en el portapapeles en el campo o variable imagen imagen.
Nota: en el caso de una operación copiar/pegar, el contenedor de datos corresponde al Portapapeles.

Ejemplo

El siguiente método de objeto de un botón asigna la imagen presente en el portapapeles (si hay) al campo [Empleados]Foto:

```
If((Pasteboard data size("com.4d.imagen.jpeg")>0)|(Pasteboard data size("com.4d.imagen.gif")>0))
   GET PICTURE FROM PASTEBOARD([Empleados]Foto)
Else
   ALERT("El portapapeles no contiene ninguna imagen.")
End if
```

Variables y conjuntos del sistema

Si la imagen se extrae correctamente, OK toma el valor 1; de lo contrario OK toma el valor 0 y se genera un error.

Get text from pasteboard

Get text from pasteboard -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Cadena*		*Devuelve el texto (si lo hay) en el Portapapeles*

Descripción

Get text from pasteboard *devuelve el texto en el portapapeles.*

Nota*: en el caso de las operaciones copiar/pegar, el contenedor de datos corresponde al Portapapeles.*

Si el contenedor de datos contiene texto enriquecido (por ejemplo en formato RTF), el texto conserva sus atributos al soltar o pegar, si el área de destino es compatible.

Note que los campos y variables de tipo texto de 4D pueden contener hasta 2 GB de texto.

Variables y conjuntos del sistema

Si el texto se extrae correctamente, OK toma el valor 1; de lo contrario OK toma el valor 0 y se genera un error.

Pasteboard data size

Pasteboard data size (tipoDatos) -> Resultado

Parámetro	Tipo		Descripción
tipoDatos	Cadena	➡	Tipo de datos (4 caracteres)
Resultado	Entero largo	➲	Tamaño (en bytes) de datos almacenados en el portapapeles o código de error

Descripción

*El comando **Pasteboard data size** permite probar si hay datos del tipo tipoDatos en el portapapeles.*

***Nota**: en el caso de las operaciones copiar/pegar, el contenedor de datos corresponde al Portapapeles*

Si el portapapeles está vacío o no contiene datos de tipo especifico, el comando devuelve un error -102 (ver la tabla de constantes predefinidas). Si el portapapeles contiene datos del tipo especificado, el comando devuelve el tamaño en bytes de estos datos.

*Pase en tipodatos un valor que defina el tipo de datos a extraer. Puede pasar una firma 4D, un tipo UTI (Mac OS), un nombre/número de formato (Windows), o un tipo de 4 caracteres (compatibilidad). Para mayor información sobre estos tipos, consulte la sección **Gestión de portapapeles**.*

Después de verificar que el portapapeles contiene datos del tipo que quiere, puede extraer esa información del portapapeles utilizando uno de los siguientes comandos:

- *Si el portapapeles contiene datos de tipo TEXT, puede obtener esa información utilizando el comando **Get text from pasteboard**, el cual devuelve un valor texto, o el comando **GET PASTEBOARD DATA**, que devuelve el texto en un BLOB.*
- *Si el portapapeles contiene datos de tipo PICT, puede obtener esos datos utilizando el comando **GET PICTURE FROM PASTEBOARD**, que devuelve la imagen en un campo o una variable o el comando **GET PASTEBOARD DATA,** que devuelve la imagen en un BLOB.*
- *Si el contenedor contiene una ruta de acceso al archivo, puede extraerla utilizando el comando pasteboard, que devuelve la ruta de acceso del archivo.*
- *Para cualquier otro tipo de datos, utilice el comando **GET PASTEBOARD DATA**, el cual devuelve los datos en un BLOB.*

Ejemplo 1

El siguiente código prueba si el portapapeles contiene una imagen y si es así, copia la imagen en una variable 4D:

```
If(Pasteboard data size(Picture data)=1) //¿Hay una imagen en el portapapeles?
   GET PICTURE FROM PASTEBOARD($vPicVariable) //Si es así, extraer la imagen del portapapeles
Else
   ALERT("No hay imagen en el portapapeles.")
End if
```

Ejemplo 2

Generalmente, las aplicaciones cortan y copian datos de tipo TEXT o PICT en el portapapeles, porque la mayoría de las aplicaciones reconocen estos dos tipos de datos estándar. Sin embargo, una aplicación puede colocar en el portapapeles varias instancias de los mismos datos en formatos diferentes. Por ejemplo, cada vez que corta o copia parte de una hoja de cálculo, la aplicación de la hoja de cálculo puede colocar los datos en un formato hipotético 'SPSH', como también en los formatos SYLK y TEXT. La instancia 'SPSH' contiene los datos estructurados en el formato utilizado por la aplicación. La copia SYLK contiene los mismos datos, pero en el formato SYLK, reconocido por la mayoría de los otros programas de hojas de cálculo. Por último, el formato TEXT contiene los mismos datos, sin la información extra incluída en el formato SYLK o en el formato hipotético 'SPSH'. En este punto, cuando escriba rutinas de Cortar/Copiar/Pegar entre 4D y una aplicación de hoja de cálculo hipotética, asumiendo que conoce la descripción del formato 'SPSH' y que está listo para analizar los datos SYLK, puede escribir el siguiente código:

```
Case of
  ` Primero, verificar si el portapapeles contiene los datos de la aplicación de la hoja de cálculo hipotética.
   :(Pasteboard data size('SPSH')>0)
  ` ...
  ` Segundo, verificar si el portapapeles contiene datos en formato Sylk
   :(Pasteboard data size('SYLK')>0)
  ` ...
  ` Por último, verificar si el portapapeles contiene datos en formato Text
   :(Pasteboard data size('TEXT')>0)
  ` ...
End case
```

En otras palabras, usted trata de extraer del portapapeles la instancia de datos que tenga más información original.

Ejemplo 3

Usted desea arrastrar algunos datos privados de diferentes objetos en su formulario. Puedes escribir:

```
//objeto origen
If(Form event=On Begin Drag Over)
   APPEND DATA TO PASTEBOARD("some.private.data";$data)
End if
```

```
//objeto destino
If(Form event=On Drag Over)
   $0:=Choose(Pasteboard data size("some.private.data")>0;0;-1)
End if
```

Ejemplo 4

*Ver el ejemplo del comando **APPEND DATA TO PASTEBOARD**.*

SET FILE TO PASTEBOARD

SET FILE TO PASTEBOARD (archivo {; *})

Parámetro	Tipo		Descripción
archivo	Cadena	➡	Nombre del archivo o ruta de acceso completa del archivo
*	Operador	➡	Si se pasa = añadir; Si se omite= reemplazar

Descripción

*El comando **SET FILE TO PASTEBOARD** añade al portapapeles la ruta de acceso completa del archivo pasada en el parámetro archivo. Este comando permite crear interfaces permitiendo arrastrar y soltar objetos 4D a los archivos en el escritorio por ejemplo.*

En el parámetro archivo, puede pasar una ruta de acceso completo o un simple nombre de archivo (sin ruta de acceso). En el último caso, el archivo debe estar ubicado junto al archivo de estructura de la base.

*El comando admite el asterisco * como parámetro opcional. Por defecto, cuando se omite este parámetro, el comando reemplaza el contenido del portapapeles por la última ruta de acceso definida por archivo. Si pasa este parámetro, el comando añade el archivo al portapapeles. De esta forma puede contener una "pila" de rutas de acceso de archivos. En ambos casos, se borran los datos diferentes a las rutas de acceso que estén en el portapapeles.*

***Nota:** el portapapeles está en modo sólo lectura durante el evento de formulario **On Drag Over**. Por lo tanto no es posible utilizar este comando en ese contexto.*

SET PICTURE TO PASTEBOARD

SET PICTURE TO PASTEBOARD (imagen)

Parámetro	Tipo		Descripción
imagen	Imagen	➡	Imagen a copiar en el portapapeles

Descripción

SET PICTURE TO PASTEBOARD *limpia el portapapeles y coloca una copia de la imagen que usted pasó en imagen en el portapapeles.*

***Nota**: en el caso de las operaciones copiar/pegar, el contenedor de datos corresponde al Portapapeles*

La imagen se pasa a su formato nativo (jpeg, tif, png, etc)

*Después de colocar la imagen en el portapapeles, puede recuperarla utilizando el comando **GET PICTURE FROM PASTEBOARD** o llamando **GET PASTEBOARD DATA("com.4d.picture.gif";...)**.*

Ejemplo

En una ventana flotante, usted visualiza un formulario que contiene el array asEmpleadoNombre, el cual lista los nombres de los empleados de una tabla [Empleados]. Cada vez que hace clic en un nombre, usted quiere copiar la imagen de un empleado en el portapapeles. En el método de objeto del Array, usted escribe:

```
If(asEmpleadoNombre#0)
   QUERY([Empleados];[Empleado]Apellido=asEmpleadoNombre{asEmpleadoNombre})
   If(Picture size([Empleados]Foto)>0)
      SET PICTURE TO PASTEBOARD([Empleados]Foto) ` Copiar la foto del empleado
   Else
      CLEAR PASTEBOARD ` No se encontró ninguna foto o registro
   End if
End if
```

Variables y conjuntos del sistema

Si una copia de la imagen se coloca correctamente en el portapapeles, la variable OK toma el valor 1.

Si no hay suficiente memoria para colocar una copia de la imagen en el portapapeles, la variable OK toma el valor 0, pero no se genera un error.

SET TEXT TO PASTEBOARD

SET TEXT TO PASTEBOARD (texto)

Parámetro	Tipo		Descripción
texto	Cadena	➡	Texto a copiar en el portapapeles

Descripción

SET TEXT TO PASTEBOARD limpia el portapapeles y luego coloca una copia del texto en texto en el portapapeles.

Nota: en el caso de las operaciones copiar/pegar, pasteboard es equivalente a clipboard.

Después de colocar texto en el portapapeles, puede recuperarlo utilizando el comando **Get text from pasteboard** o llamando **GET PASTEBOARD DATA("com.4d.text.native";...)**.

4D pueden contener hasta 2 GB de texto

Nota: el contenedor de datos está en modo sólo lectura durante el evento de formulario **On Drag Over**. No es posible utilizar este comando en este contexto.

Ejemplo

Ver el ejemplo del comando **APPEND DATA TO PASTEBOARD**.

Variables y conjuntos del sistema

Si el texto se coloca correctamente en el portapapeles, la variable OK toma el valor 1. Si no hay suficiente memoria para colocar una copia del texto en el portapapeles, la variable OK toma el valor 0, pero no se genera error.

Procesos

- *Procesos*
- *Procesos 4D apropiativos*
- *Count tasks*
- *Count user processes*
- *Count users*
- *Current process*
- *Current process name*
- *DELAY PROCESS*
- *EXECUTE ON CLIENT*
- *Execute on server*
- *Get process activity*
- *GET REGISTERED CLIENTS*
- *New process*
- *PAUSE PROCESS*
- *Process aborted*
- *Process number*
- *PROCESS PROPERTIES*
- *Process state*
- *REGISTER CLIENT*
- *RESUME PROCESS*
- *UNREGISTER CLIENT*

Procesos

Multitarea en 4D es la posibilidad de ejecutar simultáneamente varias operaciones de bases de datos distintas. Estas operaciones son llamadas procesos.

Múltiples procesos equivalen a tener múltiples usuarios trabajando en el mismo ordenador, cada uno trabajando en su propia tarea. Esto significa principalmente que un método puede ser ejecutado como una tarea distinta de base de datos.

Esta sección trata los siguientes temas:

- *Crear y eliminar procesos*
- *Elementos de un proceso*
- *Procesos creados por 4D*
- *Procesos locales y globales*
- *Bloqueo de registros entre procesos*

***Nota**: esta sección no cubre procesos almacenados. Ver la sección **Procedimientos almacenados** en el Manual de 4D Server.*

Crear y borrar procesos

Hay varias formas de crear un nuevo proceso:

- *Ejecute un método en el entorno Diseño después de seleccionar la casilla **Ejecutar en un proceso nuevo** en la caja de diálogo **Ejecutar Método**. El método elegido en la caja de diálogo Ejecutar método es el método proceso.*
- *Los procesos pueden ejecutarse seleccionando los comandos de menú. En el editor de menús, marque la casilla **Iniciar un nuevo proceso**. El método asociado al comando de menú es el método proceso.*
- *Utilice la función **New process**. El método pasado como parámetro a la función **New process** es el método proceso.*
- *Utilice la función **Execute on server** para crear un procedimiento almacenado en el servidor. El método pasado como parámetro de la función es el método proceso.*
- *Utilice el comando **CALL WORKER**. Si el proceso del worker no existe ya, se crea.*

Se puede borrar un proceso bajo las siguientes condiciones. Las primeras dos condiciones son automáticas:

- *Cuando el método proceso termina su ejecución*
- *Cuando el usuario sale de la base*
- *Si detiene el proceso con el lenguaje o utilizando el botón Abortar en el Depurador*
- *Si selecciona **Abortar** en el Explorador de ejecución*
- *Si llama al comando **KILL WORKER** (eliminar un proceso worker únicamente).*

Un proceso puede crear otro proceso. Los procesos no están organizados jerárquicamente, todos los procesos son iguales, y esto es independientemente del proceso a partir del cual ellos han sido creados. Una vez el proceso "padre" crea un proceso "hijo", el proceso hijo continua sin importar si el proceso padre se está ejecutando o no.

Elementos de un proceso

Cada proceso contiene elementos específicos. Hay tres tipos de elementos diferentes en un proceso:

*• **Elementos de interfaz**: los elementos necesarios para mostrar un proceso.*
*• **Elementos de datos**: información relacionada con los datos de la base.*
*• **Elementos de lenguaje**: elemento utilizados por el lenguaje o importantes para el desarrollo de su aplicación.*

Elementos de interfaz

Los elementos de interfaz son los siguientes:

*• **Barra de menús**: cada proceso puede tener su propia barra de menús actual. La barra de menús del proceso del primer plano es la barra de menús actual de la base.*
*• **Una o más ventanas**: cada proceso puede tener varias ventanas abiertas simultáneamente. Por el contrario, algunos procesos no tienen ventanas.*
*• **Una ventana activa (primer plano)**: aunque un proceso puede tener varias ventanas abiertas simultáneamente, cada proceso tiene sólo una ventana activa. Para tener más de una ventana activa, debe iniciar más de un proceso.*

Notas:

- *Por defecto, los procesos no incluyen barras de menús, lo que significa que los accesos directos estándar del menú **Edición** (en particular, cortar/copiar/pegar) no están disponibles en las ventanas de procesos. Cuando se llama a los diálogos o a los editores de 4D (editor de formularios, editor de búsquedas, **Request**, etc.) desde un proceso, si desea que el usuario pueda beneficiarse de atajos de teclado como copiar/pegar, debe asegurarse de que el equivalente a un menú **Edición** esté instalado en el proceso.*
- *Los procesos apropiativos y los procesos que se ejecutan en el servidor (procedimientos almacenados) no deben contener elementos de interfaz.*

Elementos de datos

Los elementos de datos se refieren a los datos de la base. Estos son los siguientes:

*• **Selección actual por tabla**: cada proceso tiene su propia selección actual. La misma tabla puede tener diferentes selecciones*

actuales en diferentes procesos.
*• **Registro actual por tabla**: cada tabla puede tener un registro actual diferente en cada proceso.*

***Nota**: esta descripción de elementos de datos es válida si los procesos son procesos globales. Por defecto, todos los procesos son globales. Consulte más adelante el párrafo que trata este tema.*

Elementos del lenguaje

Los elementos del lenguaje de un proceso son los elementos relacionados con la programación en 4D.

- ***Variables**: cada proceso tiene sus propias variables de proceso. Para mayor información ver **Variables**. Las variables proceso sólo se reconocen en el marco de su proceso nativo.*
- ***Tabla por defecto**: cada proceso tiene su propia tabla por defecto. Sin embargo, note que el comando **DEFAULT TABLE** sólo es una convención para programación.*
- ***Formularios de entrada y salida**: los formularios de entrada y salida por defecto pueden ser elegidos por programación para cada tabla en cada proceso.*
- ***Conjuntos de proceso**: cada proceso tiene sus propios conjuntos de proceso. LockedSet es un conjunto de proceso. Los conjuntos proceso se borran tan pronto como el método de proceso termina.*
- ***On Error Call por proceso**: cada proceso tiene su propio método de gestión de errores.*
- ***Ventana del depurador**: cada proceso puede tener su propia ventana del **Depurador**.*

Procesos usuario

Los procesos usuario son procesos que usted crea para efectuar ciertas tareas. Ellos comparten los tiempos de equipo con los procesos principales. Los procesos de conexión Web son los procesos usuario.

La aplicación 4D crea igualmente los procesos para sus propias necesidades. Estos son los principales procesos creados y administrados por 4D:

- ***Proceso principal**: el proceso principal administra las ventanas de visualización de la interfaz de usuario.*
- ***Proceso diseño**: el proceso desarrollo administra las ventanas y editores del entorno de Desarrollo. No hay proceso desarrollo en una base compilada que no contenga el código interpretado.*
- ***Proceso servidor web**: el proceso servidor web corre cuando la base se publica en la web. Para mayor información ver la sección .*

- ***Proceso de gestión de la cache**: el proceso de gestión de cache administra las entradas/salidas disco de la base. Este proceso se crea tan pronto como 4D o 4D Server se ejecutan.*

- ***Proceso de indexación**: el proceso de indexación administra los índices de los campos en una base como procesos por separado. Este proceso se crea cuando un índice es creado o borrado por un campo.*

- ***Proceso de gestión de eventos**: este proceso se crea cuando un método de gestión de eventos es instalado por el comando **ON EVENT CALL**. Este proceso ejecuta el método de evento instalado por **ON EVENT CALL** cuando hay un evento. El método de evento es el método de proceso de este proceso. Este proceso se ejecuta continuamente, incluso si ningún método está en ejecución. La gestión de eventos funciona también en el entorno Diseño.*

Procesos cooperativos y apropiativos

A partir de 4D v15 R5 64 bits, 4D le permite crear procesos usuario apropiativos en modo compilado. En versiones anteriores, sólo los procesos usuario cooperativos estaban disponibles.

Cuando se ejecuta en modo apropiativo, un proceso está dedicado a un CPU. La gestión de procesos a continuación, se delega al sistema, que puede asignar a cada CPU por separado en una máquina multi-núcleo. Cuando se ejecuta en modo cooperativo, todos los procesos son gestionados por el hilo de la aplicación padre y comparten la misma CPU, incluso en una máquina multi-núcleo.

Como resultado, en el modo apropiativo, el rendimiento general de la aplicación mejora, sobre todo en máquinas multi-núcleo, ya que múltiples procesos (hilos) pueden funcionar simultáneamente. Sin embargo, las ganancias reales dependen de las operaciones que se ejecuten. En contraparte, ya que cada hilo es independiente de los otros en el modo apropiativo, y no gestionado directamente por la aplicación, existen limitaciones específicas aplicadas a los métodos que se desea para ser compatibles con el uso apropiativo. Además, la ejecución preferente sólo está disponible en ciertos contextos específicos.

*La gestión de los procesos apropiativos se detalla en la sección **Procesos 4D apropiativos**.*

Procesos globales y locales

El alcance de los procesos puede ser local o global. Por defecto, todos los procesos son globales.

En la mayoría de los casos, usted utilizará procesos globales. Los procesos globales pueden efectuar toda operación, incluyendo acceso y manipulación de los datos.

Los procesos locales deben ser utilizados únicamente para operaciones que no acceden a los datos. Por ejemplo, puede utilizar un proceso local para controlar los elementos de interfaz como las paletas flotantes o ejecutar un método de gestión de eventos.

***Atención**: si intenta acceder a los datos desde un proceso local, accede a los datos por medio del proceso principal, y toma el riesgo de entrar en conflicto con las operaciones efectuadas en ese proceso.*

Se especifica que un proceso es local por medio de su nombre. El nombre de un proceso local debe comenzar por el símbolo dólar.

***4D Server**: con 4D Server, el uso de procesos locales por parte del cliente para operaciones que no requieren accesos a los datos permite asignar aún más tiempo de procesamiento a tareas que requieren el servidor intensivamente.*

Bloqueo de registros entre procesos

Un registro está bloqueado cuando otro proceso ha cargado el registro para modificarlo. Un registro bloqueado puede ser cargado por otro proceso, pero no puede ser modificado. El registro está bloqueado solamente en el proceso en el cual el registro

está siendo modificado. Una tabla debe estar en modo lectura/escritura para que un registro se cargue desbloqueado. Para mayor información consulte la sección .

Procesos 4D apropiativos

*4D Developer Edition 64 bits para Windows y OS X ofrecen una poderosa funcionalidad: la posibilidad de escribir y ejecutar código **4D apropiativo**. Gracias a esta nueva funcionalidad, sus aplicaciones 4D compiladas podrán sacar el máximo provecho de los ordenadores de varios núcleos de modo que su ejecución será más rápida y puede soportar más usuarios conectados.*

¿Qué es un proceso apropiativo?

Cuando se ejecuta en modo apropiativo, un proceso está dedicado a una CPU. La gestión de procesos luego se delega al sistema, que puede adjudicar por separado cada CPU en una máquina multi-núcleo.

Cuando se ejecuta en modo cooperativo (el único modo disponible en 4D hasta 4D v15 R5), todos los procesos son gestionados por el hilo de la aplicación padre y comparten la misma CPU, incluso en una máquina multi-núcleo.

Como resultado, en el modo apropiativo, el rendimiento global de la aplicación se incrementa, sobre todo en máquinas multi-núcleo, ya que múltiples procesos (hilos) se pueden ejecutar simultáneamente. Sin embargo, las ganancias reales dependen de las operaciones ejecutadas.

En contraparte, ya que en el modo apropiativo cada hilo es independiente de los demás y no es gestionado directamente por la aplicación, limitaciones específicas se aplican a los métodos que usted desea que sean compatibles con el modo apropiativo. Además, la ejecución apropiativa está disponible sólo en algunos contextos específicos.

Disponibilidad del modo apropiativo

*El uso del modo apropiativo está disponible versiones **4D 64 bits únicamente**. Los siguientes contextos de ejecución son actualmente soportados:*

	Ejecución apropiativa
4D Server	*X*
4D remoto	*-*
4D mono usuario	*X*
Modo compilado	*X*
Modo interpretado	*-*

*Si el contexto de ejecución soporta el modo apropiativo y si el método es "hilo de seguro", un proceso 4D lanzado utilizando los comandos **New process** o **CALL WORKER**, o vía el comando de menú "Ejecutar método" se ejecutará en modo apropiativo. De lo contrario, si llama **New process** o **CALL WORKER** desde un contexto de ejecución que no es soportado (por ejemplo en una máquina 4D remota), el proceso es siempre cooperativo.*

***Nota**: puede ejecutar un proceso en modo apropiativo desde un 4D remoto iniciando un procedimiento almacenado en el servidor con el lenguaje, por ejemplo, con **Execute on server**.*

Hilo seguro vs hilo inseguro

*El código 4D se puede ejecutar en hilo apropiativo sólo cuando se cumplen algunas condiciones específicas. Cada parte del código ejecutado (comandos, métodos, variables...) debe ser compatible con una ejecución apropiativa. Los elementos que se pueden ejecutar en hilos apropiativos se llaman **hilos seguros** y los elementos que no se pueden ejecutar en hilos apropiativos se llaman **hilos inseguros**.*

***Nota:** dado que un hilo se maneja de forma independiente a partir del método proceso padre, toda la cadena de llamadas no debe incluir ningún código hilo inseguro, de lo contrario la ejecución apropiativa no será posible. Este punto se discute en el párrafo **¿Cuando un proceso se inicia apropiativamente?**.*

La propiedad "seguridad de hilo" de cada elemento depende del elemento en sí:

- *Comandos 4D: hilo seguro es una propiedad interna. En la referencia del lenguaje, los comandos hilo seguro se identifican por la imagen  . Una gran parte de los comandos 4D pueden ejecutarse en modo apropiativo.*
- *Métodos de proyecto: las condiciones para seguridad de hilo se listan en el párrafo **Escribir un método hilo seguro**.*

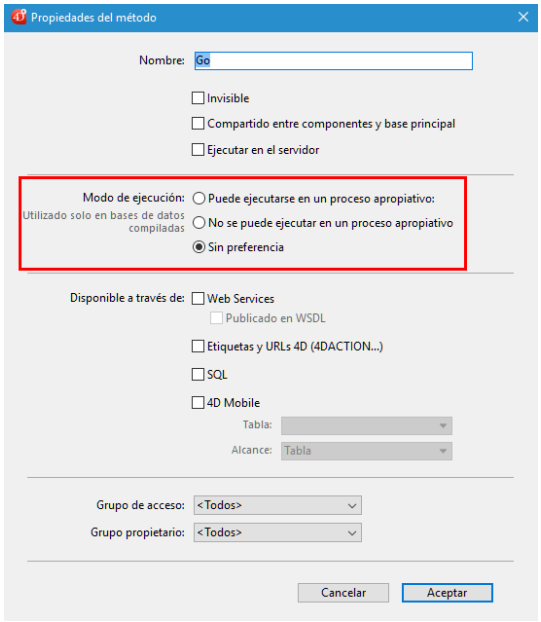
Básicamente, el código que se ejecuta en hilos apropiativos no puede llamar a las partes con las interacciones externas, tal como el código plug-in o las variables interproceso. Los accesos a los datos, sin embargo, son permitidos desde el servidor de datos 4D que soporta la ejecución apropiativa.

Declaración del modo ejecución apropiativo

*Por defecto, 4D ejecuta todos los métodos de proyecto de su aplicación en modo cooperativo. Si desea beneficiarse de la funcionalidad modo apropiativo, el primer paso consiste en declarar explícitamente todos los métodos que desea que se inicien en modo apropiativo siempre que sea posible, es decir, los métodos que considere capaz de ejecutar en proceso apropiativo. El compilador comprobará que estos métodos sean en realidad hilo seguro (ver **Escribir un método hilo seguro** para más información). También puede inhabilitar el modo apropiativo para algunos métodos, si es necesario.*

*Tenga en cuenta que la definir un método como apropiativo hace que sea elegible para ejecución apropiativa, pero no garantiza que se ejecute realmente en modo apropiativo. Iniciar un proceso en modo apropiativo resulta de una evaluación realizada por 4D respecto a las propiedades de todos los métodos en la cadena de llamadas del proceso (para más información, consulte el párrafo **¿Cuando un proceso se inicia apropiativamente?**.*

Para declarar su método de elegibilidad para el modo apropiativo, es necesario utilizar la opción de declaración ***Modo de ejecución*** *en el cuadro de diálogo Propiedades del método:*



Están disponibles las siguientes opciones:

- ***Puede ejecutarse en un proceso apropiativo****: al seleccionar esta opción, declara que el método es capaz de ejecutarse en un proceso apropiativo y por lo tanto se debe ejecutar en modo apropiativo cuando sea posible. La propiedad "apropiativa" del método se activa.*
  Cuando se selecciona esta opción, el compilador 4D verificará que el método esté realmente activo y devolverá error si no es el caso, por ejemplo, si directa o indirectamente llama a comandos o métodos que no se pueden ejecutar en modo apropiativo (toda la cadena de llamadas se analiza, pero sólo los errores son reportados al primer subnivel). Luego, puede editar el método para que sea hilo seguro o seleccionar otra opción.
  Si la elegibilidad del método al modo apropiativo es aprobada, se etiqueta "hilo-seguro" internamente y se ejecutará en el modo apropiativo siempre que se cumplan las condiciones requeridas. Esta propiedad define su elegibilidad para el modo apropiativo, pero no garantiza que el método realmente se puede ejecutar en modo apropiativo, ya que este modo de ejecución requiere un contexto específico (ver ***¿Cuando un proceso se inicia apropiativamente?****).*

- ***No se puede ejecutar en un proceso apropiativo****: seleccionando esta opción, se declara que el método no debe ejecutarse en modo apropiativo, y por lo tanto siempre se debe ejecutar en modo cooperativo, al igual que en las versiones anteriores de 4D. La propiedad "apropiativa" del método se activa.*
  Cuando se selecciona esta opción, el compilador 4D no verificará la capacidad del método para ejecutarse de forma apropiativa; se marca de forma automática "hilo-inseguro" internamente (incluso si es teóricamente compatible). Cuando se llama en ejecución, este método "contamina" cualquier otro método en el mismo hilo, lo que obliga a este hilo a ser ejecutado en modo cooperativo, incluso si los otros métodos son hilo seguro.

- ***Indiferente*** *(por defecto): al seleccionar esta opción, usted declara que no desea manejar la propiedad apropiativa para el método. La propiedad "apropiativa" del método se establece como "indiferente".*
  Cuando se selecciona esta opción, el compilador 4D evaluará la capacidad apropiativa del método y la etiquetará internamente como "hilo seguro " o "hilo inseguro". No se devuelve ningún error relacionado con la ejecución apropiativa. Si se evalúa el método como hilo seguro, la ejecución no impedirá la utilización del modo apropiatvo si se llama en un contexto apropiativo. Por el contrario, si el método se evalúa "hilo inseguro", durante la ejecución se evitará cualquier utilización del modo apropiativo cuando se le llame.
  Tenga en cuenta que con esta opción, cualquiera que sea la evaluación interna de seguridad de hilo, siempre se ejecutará el método en modo cooperativo cuando sea llamado directamente por 4D como el primer método array (por ejemplo a través del comando ***New process****). Si se ha etiquetado como "hilo seguro" internamente, sólo se tendrá en cuenta cuando se llama desde otros métodos dentro de una cadena de llamadas.*

Nota: *un método de componentes declarados como "Compartido con componentes y bases locales" también debe ser declarado "capaz" con el fin de que pueda correr en un hilo apropiativo por la base local.*

Al exportar el código del método utilizando, por ejemplo ***METHOD GET CODE****, la propiedad "apropiativa" se exporta en el comentario atributo con valor "auto", "seguro", o "inseguro". Los comandos* ***METHOD GET ATTRIBUTES*** *y* ***METHOD SET ATTRIBUTES*** *también obtienen o establecen el atributo "apropiativo" con valor "auto", "seguro", o "inseguro".*

La siguiente tabla resume los efectos de las opciones de declaración del modo apropiativo:

Opción	Valor de la propiedad apropiativa (interpretado)	Acción compilador	Etiqueta interna (compilada)	Modo de ejecución, si la cadena de llamadas es hilo seguro
Se puede ejecutar en procesos apropiativos	*capaz*	*Comprobar la capacidad y devuelve errores si fuera incapaz*	*Hilo seguro*	*Apropiativo*
No se puede ejecutar en los procesos apropiativos	*incapaz*	*Sin evaluación*	*Hilo inseguro*	*Cooperativo*
Indiferente	*indiferente*	*Evaluación, pero no hay errores devueltos*	*hilo de seguridad o hilo inseguro*	*Si hilo seguro: apropiativo; Si hilo inseguro: cooperativo; si es llamado directamente: cooperativo*

¿Cuando un proceso se inicia apropiativamente?

Recordatorio: *la ejecución apropiativa sólo está disponible en modo compilado.*

En modo compilado, cuando se inicia un proceso creado por los métodos ***New process*** *o* ***CALL WORKER****, 4D lee la propiedad apropiativa del método proceso (también llamado método padre) y ejecuta el proceso en modo apropiativo o cooperativo, en función de esta propiedad:*

- *Si el método proceso es hilo seguro (validado durante la compilación), el proceso se ejecuta en un hilo apropiativo.*
- *Si el método proceso hilo inseguro, el proceso se ejecuta en un hilo cooperativo.*
- *Si la propiedad apropiativa del método de proceso era "indiferente", por compatibilidad el proceso se ejecuta en un hilo cooperativo (incluso si el método es realmente capaz). Note sin embargo que esta funcionalidad de compatibilidad se aplica sólo cuando se utiliza el método como método de proceso: un método declarado "indiferente", pero internamente etiquetado "hilo seguro" por el compilador se puede llamar de forma apropiativa por otro método (ver abajo).*

La propiedad seguridad de hilo depende de la cadena de llamadas. Si un método con la propiedad declarada "capaz" llama a un método hilo inseguro en cualquiera de sus subniveles, se devolverá un error de compilación: si un método único en toda la cadena de llamadas es hilo inseguro, "contaminará" todos los otros métodos y la ejecución apropiativa será rechazada por el compilador. Un hilo apropiativo sólo puede crearse cuando toda la cadena es hilo seguro y el método de proceso ha sido declarado "Se puede ejecutar en proceso apropiativo".
Por otra parte, un mismo método hilo seguro puede ejecutarse en un hilo apropiativo en una cadena de llamada y en hilo cooperativo en otra cadena de llamada.

Por ejemplo, considere los siguientes métodos de proyecto:

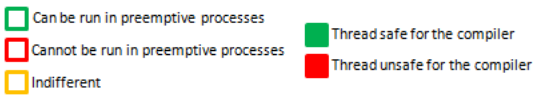
```
//Método proyecto MyDialog
//contiene llamadas de interfaz: será hilo no seguro internamente
$win:=Open window("tools";Palette form window)
DIALOG("tools")
```

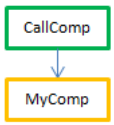
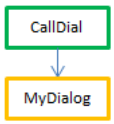
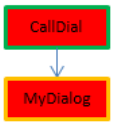
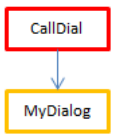
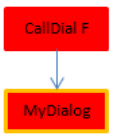
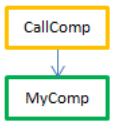
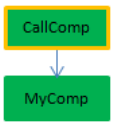
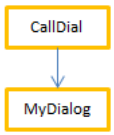
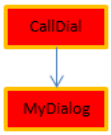
```
//Método proyecto MyComp
//contiene calculo simple: será hilo seguro internamente
C_LONGINT($1)
$0:=$1*2
```

```
//Método proyecto CallDial
C_TEXT($vName)
MyDialog
```

```
//Método proyecto CallComp
C_LONGINT($vAge)
MyCom($vAge)
```

La ejecución de un método en el modo apropiativo dependerá de que la propiedad "ejecución" y de la cadena de llamadas. La siguiente tabla ilustra estas diversas situaciones:



Declaración y cadena de llamadas	Compilación	Seguridad de hilo resultante	Comentario	Ejecución
	*OK*		*Método A es el método padre, declarado "capaz" de uso apropiativo; ya que el método B es internamente hilo seguro, el método A es hilo seguro y el proceso es apropiativo*	*Apropiativo*
	*Error*		*El método C es el método padre, declarado "capaz"; sin embargo, ya que el método E está internamente hilo inseguro, contamina la cadena de llamadas. La compilación falla debido a un conflicto entre la declaración del método C y la capacidad real*	*La solución es o bien modificar el método E para que sea hilo seguro (suponiendo que el método D es hilo seguro), de modo que la ejecución es apropiativa, o para cambiar la declaración de propiedad del método C a cooperativo*
	*OK*		*Como el método F se declara "incapaz" de uso apropiativo, la compilación es internamente hilo inseguro, la ejecución siempre será cooperativa, cualquiera que sea el estado del método G*	*Cooperativo*
	*OK*		*Desde que el método H es el método padre (la propiedad era "indiferente") , el proceso es cooperativo. La compilación es exitosa, incluso si el método fue declarado "capaz"*	*Cooperativo*
	*OK*		*el método J es el método padre (propiedad "indiferente"), entonces el proceso es cooperativo, incluso si toda la cadena es hilo seguro*	*Cooperativo*

Cómo buscar el modo de ejecución actual

4D le permite identificar la ejecución de procesos en modo compilado:

- *El comando **PROCESS PROPERTIES** le permite averiguar si un proceso se ejecuta en modo apropiativo o cooperativo.*
- *El Explorador de ejecución y la ventana de administración de 4D Server muestran nuevos iconos específicos para los procesos apropiativos (así como también para procesos worker):*

Tipo de proceso	Icono
Proceso almacenado apropiativo	
Proceso worker apropiativo	
Proceso worker cooperativo	

Escribir un método hilo seguro

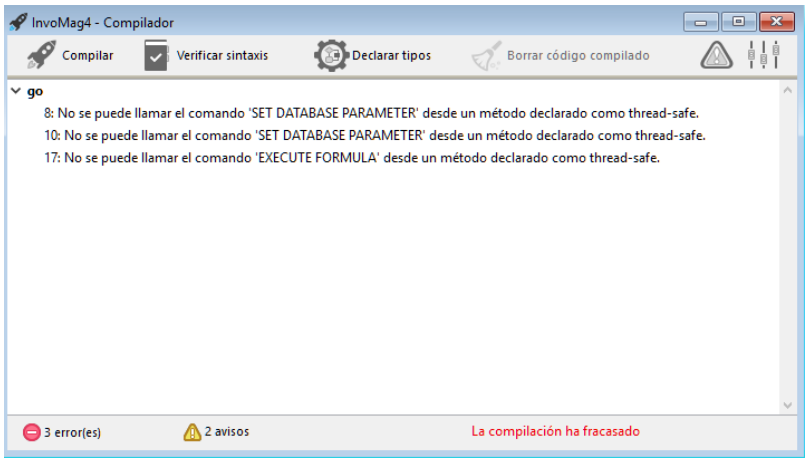
Para ser hilo seguro, un método debe respetar las siguientes reglas:

- *debe tener la propiedad "Se puede ejecutar en procesos apropiativos" o "Indiferente"*
- *no llama a un comando 4D que es hilo inseguro.*
- *no llama a otro método de proyecto que es hilo inseguro*
- *no llama a un plug-in*
- *no utiliza bloques de código **begin sql/end sql***
- *no utiliza ninguna variable interprocesos(*)*
- *no llama a objetos de interfaz(**) (hay excepciones, sin embargo, ver más adelante).*

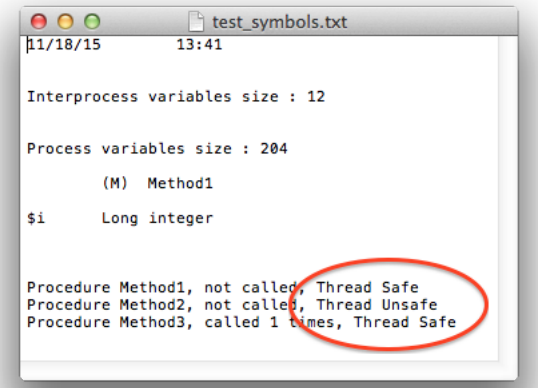
***Nota:** en el caso de un método "Compartido por componentes y bases locales", la propiedad "Se puede ejecutar en procesos apropiativos" debe ser seleccionada.*

*(*) Para intercambiar datos entre procesos apropiativos (y entre todos los procesos), puede pasar colecciones compartidas u objetos compartidos como parámetros a procesos, y/o usar el catálogo **Storage**. Para obtener más información, consulte la página **Objetos y colecciones compartidos**.*
*Los procesos worker también le permiten intercambiar mensajes entre procesos, incluidos los procesos apropiativos. Para más información, consulte el **Sobre workers**.*
*(**) El comando **ClientComment** ofrece una solución elegante para llamar a objetos de interfaz de un proceso apropiativo.*

Los métodos con la propiedad "Se puede ejecutar en procesos apropiativos" serán verificados por 4D en el paso de compilación. Un error de compilación se emite cuando el compilador encuentra algo que le impide ser hilo seguro:



El archivo de símbolos, si está habilitado, también contiene el estado de hilo de seguridad para cada método:



Interfaz

Ya que son accesos "externos", las llamadas a objetos de interfaz de usuario, tales como formularios, así como también al Depurador no están permitidas en hilos apropiativos.

Los únicos accesos posibles a la interfaz de usuario de un hilo apropiativo son:

- *diálogo de error estándar. El diálogo se muestra en el proceso modo de usuario (en 4D monousuario) o el proceso de interfaz de usuario del servidor (4D Server). El botón **Rastreo** está inhabilitado.*
- *indicadores de progreso estándar*
- *Diálogos ALERT, REQUEST y CONFIRM. El diálogo se muestra en el proceso modo usuario (en 4D monousuario) o el proceso de interfaz de usuario del servidor (4D Server).*
  Tenga en cuenta que si 4D Server se ha lanzado como un servicio en Windows sin intervención del usuario permitido, no se mostrarán los diálogos.

Comandos 4D hilo seguro

Un número significativo de comandos 4D son hilo seguro. En la documentación, la imagen  *en el área de propiedad del comando indica que el comando es hilo seguro. Puede obtener la lista de comandos hilo seguro en el manual Referencia del lenguaje.*

*También puede utilizar **Command name** que puede devolver la propiedad hilo seguro para cada comando (ver abajo).*

Triggers

Cuando un método utiliza un comando que puede llamar un disparador, el compilador 4D evalúa la seguridad de hilo del disparador para comprobar la seguridad de hilo del método:

```
SAVE RECORD([Table_1]) //dispara en Table_1, si existe, debe ser hilo seguro
```

Esta es la lista de comandos que se verifican en el momento de la compilación para la seguridad de hilo de los disparadores:

- *SAVE RECORD*
- *SAVE RELATED ONE*
- *DELETE RECORD*
- *DELETE SELECTION*
- *ARRAY TO SELECTION*
- *JSON TO SELECTION*
- *APPLY TO SELECTION*
- *IMPORT DATA*
- *IMPORT DIF*
- *IMPORT ODBC*
- *IMPORT SYLK*

- *IMPORT TEXT*

Si la tabla se pasa dinamicamente, el compilador puede algunas veces no encontrar que trigger evaluar. Estos son algunos ejemplos de cada situación:

```
DEFAULT TABLE([Table_1])
SAVE RECORD
SAVE RECORD($ptrOnTable->)
SAVE RECORD(Table(myMethodThatReturnsATableNumber())->)
```

En este caso, todos los triggers son seleccionados.Si se detecta un comando hilo inseguro en al menos un trigger, todo el grupo se rechaza y el método se declara hilo inseguro.

Métodos de captura de errores

*Los métodos de captura de errores instalados por el comando **ON ERR CALL** deben ser hilo seguro si es probable que sean llamados desde un proceso apropiativo. Con el fin de manejar este caso, el compilador verifica la propiedad de seguridad de hilo de los métodos de proyecto de captura de errores pasados al comando **ON ERR CALL** durante la compilación y devuelve los errores correspondientes si no cumplen con la ejecución apropiativa..*

Tenga en cuenta que esta comprobación sólo es posible cuando el nombre del método se pasa como una constante, y no se calcula, como se muestra a continuación:

```
ON ERR CALL("myErrMethod1") //será verificado por el compilador
ON ERR CALL("myErrMethod"+String($vNum)) //no será verificado por el compilador
```

Además, a partir de 4D v15 R5, si un método de proyecto de captura de errores no se puede llamar en tiempo de ejecución (luego de un problema de seguridad hilo, o por cualquier razón como "método no encontrado"), se genera el nuevo error -10532 "No se puede llamar al método de proyecto de gestión de errores 'methodName'".

Compatibilidad de punteros

Un proceso puede desreferenciar a un puntero para acceder al valor de otra variable proceso sólo si ambos procesos son cooperativos, de lo contrario 4D generará un error. En un proceso apropiativo, si algún código 4D intenta desreferenciar un puntero a una variable interproceso, 4D generará un error.

Ejemplo con los siguientes métodos:

Method1:

```
myVar:=42
$pid:=New process("Method2";0;"process name";->myVar)
```

Method2:

```
$value:=$1->
```

Si el proceso que ejecuta Method1 o el proceso que ejecuta Method2 es apropiativo, luego la expresión "$value:=$1->" lanzará un error de ejecución.

Referencia de documento refDoc

*El uso de parámetros de tipo DocRef (referencia de documento abierto, utilizado o devuelto por los comandos **Open document**, **Create document**, **Append document**, **CLOSE DOCUMENT**, **RECEIVE PACKET**, **SEND PACKET**) se limita a los siguientes contextos:*

- *Cuando se llama desde un proceso apropiativo, una referencia DocRef sólo es utilizable a partir de ese proceso apropiativo.*
- *Cuando se llama desde un proceso cooperativo, una referencia DocRef se puede utilizar desde cualquier otro proceso cooperativo.*

*Para más información sobre la referencia DocRef, consulte la sección **DocRef: número de referencia del documento**.*

Count tasks

Count tasks -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo	➲	Número total de procesos abiertos (incluyendo procesos kernel)

Descripción

Count tasks *devuelve el número de procesos vivos más alto en un equipo 4D remoto, 4D 4D Server (procedimientos almacenados) o en una versión monousuario de 4D. Los procesos se numeran en el orden en que se crean. Cuando aún no se ha interrumpido ningún proceso durante la sesión, este comando devuelve el número total de procesos abiertos.*
Este número tiene en cuenta todos los procesos, incluso aquellos administrados automáticamente por 4D. Esto incluye el proceso principal, la gestión de la caché, el proceso de diseño, la gestión de índices o el proceso del servidor Web.

Ejemplo

Ver el ejemplo para ***Process state*** *y On Exit Database Method.*

Count user processes

Count user processes -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entero largo*		*Número de procesos abiertos (a excepción de los procesos kernel)*

Descripción

Count user processes *devuelve el número de los procesos abiertos directa o indirectamente por el usuario (procesos para los cuales el parámetro origen devuelto por el comando* ***Count user processes*** *es mayor o igual a 0).*

Count users

Count users -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entero largo*		*Número de usuarios conectados al servidor*

Descripción

Cuando se llama desde un procedimiento almacenado en el servidor, el comando ***Count users*** *devuelve el número de usuarios conectados al equipo servidor.*

Si el servidor está ejecutando al menos un procedimiento almacenado y si ***Count users*** *se llama desde otro contexto (equipo cliente, método web), el comando devuelve el número de usuarios +1.*

En versión monousuario de 4D, ***Count users*** *devuelve 1.*

Current process

Current process -> Resultado

Parámetro	***Tipo***	***Descripción***
Resultado	*Entero largo*	*Número de proceso*

Descripción

Current process *devuelve el número de proceso a partir del cual se llama este comando.*

Ejemplo

Ver los ejemplos de ***DELAY PROCESS*** *y* ***PROCESS PROPERTIES****.*

Current process name

Current process name -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Texto*	➲	*Nombre del proceso actual*

Descripción

*El comando **Current process name** devuelve el nombre del proceso en el que este comando se llama.*
*Este comando es particularmente útil en el contexto de los procesos de trabajo (ver la sección **Sobre workers**). Puede ser utilizado para identificar el proceso worker a llamar al escribir código genérico.*

Ejemplo

Quiere llamar a un proceso worker y pasar el nombre del proceso de llamada como parámetro:

```
CALL WORKER(1;"myMessage";Current process name;"Start:"+String(vMax))
```

DELAY PROCESS

DELAY PROCESS (proceso ; duracion)

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número de proceso
duracion	Real	➡	Duración expresada en tics

Descripción

DELAY PROCESS *permite retrasar un proceso por un número de tics (1 tic = 1/60 de segundo). Durante este periodo, el process retrasado no utiliza tiempo de procesamiento. Aunque la ejecución de un proceso puede ser retardada, aún está en memoria.*

Puede retrasar un proceso por menos de un tic. Por ejemplo, si pasa 0.5 en duración, el proceso será retrasado por 1/2 tic, es decir 1/120 de segundo.

Si el proceso ya está retrasado, este comando lo retrasa nuevamente. El parámetro duracion no se añade al tiempo restante, pero lo reemplaza. Por lo tanto pase cero (0) en duracion si no quiere retrasar el proceso.

Si el proceso no existe, el comando no hace nada.

Nota: *no puede utilizar este comando para asignar un procedimiento almacenado en el equipo servidor de un equipo cliente (process<0).*

Ejemplo 1

Ver los ejemplos en ***Record Locking****.*

Ejemplo 2

Ver el ejemplo del comando ***Process Number****.*

EXECUTE ON CLIENT

EXECUTE ON CLIENT (nomCliente ; nomMetodo {; param}{; param2 ; ... ; paramN})

Parámetro	Tipo		Descripción
nomCliente	Cadena	➡	Nombre registrado de 4D Client
nomMetodo	Cadena	➡	Nombre del método a ejecutar
param		➡	Parámetro(s) del método

Descripción

*El comando **EXECUTE ON CLIENT** provoca la ejecución del método nomMetodo, con los parámetros param1... paramN, si es necesario, en el cliente 4D registrado cuyo nombre es nomCliente. El nombre registrado del cliente 4D es definido por el comando **REGISTER CLIENT**.*
Este comando puede llamarse desde un cliente 4D o desde un procedimiento almacenado en 4D Server.

Si el método requiere uno o más parámetros, páselos después del nombre del método.
La ejecución del método en el cliente 4D se efectúa en un proceso creado automáticamente en el equipo cliente, y su nombre será el nombre de registro de 4D Client.

*Si este comando se llama muchas veces para un mismo 4D Client, las órdenes de ejecución serán apiladas. Por lo tanto, los métodos serán tratados uno después del otro en modo asincrónico. Entre más métodos estén apilados, mayor será la carga de trabajo para el 4D Client. Ahora usted puede conocer el estado de la carga de trabajo de cada cliente utilizando el comando **GET REGISTERED CLIENTS**.*

***Nota:** el apilamiento de órdenes de ejecución no puede ser modificado o detenido, a menos que el cliente 4D se saque del registro con la ayuda del comando **UNREGISTER CLIENT**.*

Puede ejecutar simultáneamente el mismo método en varios o en todos los 4D Clients registrados. Para hacerlo, utilice el carácter arroba (@) en el parámetro nomCliente.

Ejemplo 1

Asumamos que quiere ejecutar el método "GenerarNums" en el equipo cliente "Client1":

```
EXECUTE ON CLIENT("Client1";"GenerarNums";12;$a;"Text")
```

Ejemplo 2

Si quiere que todos los clientes ejecuten el método "VacioTiemp":

```
EXECUTE ON CLIENT("@";"EmptyTemp")
```

Ejemplo 3

*Consulte el ejemplo del comando **REGISTER CLIENT**.*

Variables y conjuntos del sistema

*La variable sistema **OK** es igual a 1 si 4D Server ha recibido correctamente petición de ejecución de un método; sin embargo, esto no garantiza que el método ha sido ejecutado correctamente por 4D Client.*

Execute on server

Execute on server (proced ; pila {; nombre {; param {; param2 ; ... ; paramN}}}{; *}) -> Resultado

Parámetro	Tipo		Descripción
proced	Cadena	➡	Procedimiento a ejecutar en el proceso
pila	Entero largo	➡	Tamaño de la pila en bytes
nombre	Cadena	➡	Nombre del proceso creado
param	Expresión	➡	Paramétro(s) del procedimiento
*	Operador	➡	Proceso único
Resultado	Entero largo	⮌	Número de proceso para el proceso creado recientemente o de un proceso que está siendo ejecutado

Descripción

*El comando **Execute on server** inicia un nuevo proceso en el equipo servidor (cuando se llama en Cliente/Servidor) o en el mismo equipo (si se llama en monousuario) y devuelve el número de este proceso.*

*Utilice **Execute on server** para iniciar un procedimiento almacenado. Para mayor información sobre procedimientos almacenados, consulte la sección **Stored Procedures** en el manual de referencia de 4D Server.*

*Si llama **Execute on server** en un equipo cliente, el comando devuelve un número de proceso negativo. Si llama **Execute on server** en el equipo servidor, **Execute on server** devuelve un número de proceso positivo. Note que llamar **New process** en el equipo servidor hace lo mismo que llamar **Execute on server**.*

*Si el proceso no se pudo crear (por ejemplo, si no hay suficiente memoria), **Execute on server** devuelve (0) y se genera un error. Puede interceptar este error utilizando un método de gestión de errores instalado por el comando **ON ERR CALL**.*

Método de proceso

En proced, pase el nombre del método de proceso para el nuevo proceso. Una vez 4D haya definido el contexto para el nuevo proceso, comienza la ejecución de este método, el cual se convierte en el método de proceso.

Pila de proceso

El parámetro pila, permite indicar la cantidad de memoria asignada para la pila del proceso. Este valor representa el espacio en memoria utilizado para "apilar" las llamadas de métodos, las variables locales, los parámetros de subrutinas y los registros apilados.

- *Pase 0 en pila para utilizar un tamaño de pila por defecto, conveniente para la mayoría de las aplicaciones (configuración recomendada).*
- *En algunos casos particulares, puede querer un valor personalizado. Se expresa en bytes; se recomienda pasar al menos 64K (alrededor de 64 000 bytes) y puede utilizar valores sobre 512 KB en particular si los procesos pueden realizar llamadas a cadenas largas (subrutinas llamando subrutinas en cascada).*

***Nota:** la pila NO es la memoria total reservada para el proceso. Los procesos comparten memoria para los registros, las variables interproceso, etc. Un proceso utiliza igualmente la memoria extra para almacenar sus variables proceso. La pila contiene diferente información interna 4D; la cantidad de información guardada en la pila depende del número de llamadas a métodos anidados que el proceso emplee, el número de formularios que abrirá antes de cerrarlos y el número y tamaño de las variables locales utilizadas en cada llamada al método anidado.*

***Nota 4D Server 64 bits**: la pila de un proceso ejecutado en 4D Server 64 bits requiere más memoria que en 4D Server 32 bits (alrededor del doble). Asegúrese de verificar este parámetro cuando su código esté destinado a ser ejecutado en 4D Server 64 bits.*

Nombre del proceso

*El nombre del nuevo proceso se pasa en nombre. En monousuario, este nombre aparecerá en la lista de procesos del entorno Diseño, y será devuelto por el comando **PROCESS PROPERTIES** cuando se aplica a este nuevo proceso. En cliente/servidor, este nombre aparecerá en azul en la lista de **Procedimientos almacenado**s de la ventana principal de 4D Server.*

Puede omitir este parámetro; si lo hace, el nombre del proceso será una cadena vacía.

***Advertencia:** a diferencia del comando **Nuevo Proceso**, no intente crear un proceso local colocándole el signo dólares ($) como prefijo al nombre mientras utiliza **Execute on server**. Funcionará correctamente en monousuario, porque **Execute on server** actúa como **New Process** en este entorno. Por otra parte, en Cliente/Servidor, esto generará un error.*

Parámetros del método proceso

Puede pasar parámetros al método de proceso. Puede pasar parámetros de la misma forma como los pasaría a una subrutina. Sin embargo, hay una restricción, no puede pasar expresiones de tipo puntero. Igualmente, recuerde que los arrays no pueden ser pasados como parámetros a un método. Una vez comience la ejecución en el contexto del nuevo proceso, el método de proceso recibe los valores de los parámetros en $1, $2, etc.

***Nota:** si pasa parámetros al método de proceso, debe pasar el parámetro nombre; en este caso no se puede omitir.*
*Si pasa un objeto (**C_OBJECT**) o una colección (**C_COLLECTION**) como param, se envía una copia (y no una referencia) y el formulario JSON se utiliza en UTF-8 para el servidor. Si el objeto o colección contiene punteros, se envían sus valores desreferenciados y no los punteros mismos.*

Parámetro opcional *

Si pasa este último parámetro le pide a 4D verificar primero si se está ejecutando un proceso con el mismo nombre que pasó en nombre. Si es así, 4D no inicia un nuevo proceso y devuelve el número del proceso con ese nombre.

Ejemplo

*El siguiente ejemplo muestra cómo la importación de datos puede acelerarse de manera dramática en entorno Cliente/Servidor. El método **Importacion Clasica** listado a continuación le permite medir cuánto tiempo toma una importación de registros utilizando el comando **IMPORT TEXT**:*

```
  `Método de proyecto Importacion Clasica
$vhDocRef:=Open document("")
If(OK=1)
   CLOSE DOCUMENT($vhDocRef)
   FORM SET INPUT([Tabla1];"Import")
   $vhHoraInicial:=Current time
   IMPORT TEXT([Tabla1];Document)
   $vhHoraFinal:=Current time
   ALERT("La operación tardó "+String(0+($vhHoraFinal-$vhHoraInicial))+" segundos.")
End if
```

*Con la importación de datos clásica, 4D Client analiza el archivo ASCII, luego, para cada registro, crea un nuevo registro, llena los campos con los datos importados y envía el registro al equipo servidor de manera que pueda ser añadido a la base. Por lo tanto hay muchas peticiones circulando por la red. Una manera de optimizar la operación es utilizar un procedimiento almacenado para hacer el trabajo localmente en el equipo servidor. El equipo cliente carga el documento en un BLOB, comienza un procedimiento almacenado pasando el BLOB como parámetro. El procedimiento almacenado coloca el BLOB en un documento en el disco del equipo servidor, luego importa el documento localmente. Por lo tanto, la importación de datos se lleva a cabo localmente a una velocidad comparable a la de una versión monopuesto de 4D, porque la mayoría de las peticiones que circulan por la red han sido eliminadas. Este es el método de proyecto **CLIENTE IMPORTAR**. Ejecutado en el equipo cliente, inicia la ejecución del procedimiento almacenado **SERVIDOR IMPORTAR,** listado a continuación:*

```
  ` Método de proyecto CLIENTE IMPORTAR
  ` CLIENTE IMPORTAR ( Puntero ; Alfa )
  ` CLIENTE IMPORTAR ( -> [Tabla] ; Formulario de entrada )

C_POINTER($1)
C_TEXT($2)
C_TIME($vhDocRef)
C_BLOB($vxData)
C_LONGINT(spErrCode)

  ` Seleccione el documento a importar
$vhDocRef:=Open document("")
If(OK=1)
  ` Si un documento fue seleccionado, no lo deje abierto
   CLOSE DOCUMENT($vhDocRef)
   $vhHoraInicial:=Current time
  ` Trate cargarlo en memoria
   DOCUMENT TO BLOB(Document;$vxData)
   If(OK=1)
  ` Si el documento puede ser cargado en el BLOB,
  ` Inicie el procedimiento almacenado que importará los datos en el equipo servidor
      $spProcessID:=Execute on server("SERVIDOR IMPORTAR";0;"Servidor Importando";Table($1);$2;$vxData)
  ` En este punto, ya no necesitamos más el BLOB en este proceso
      CLEAR VARIABLE($vxData)
  ` Espere la terminación de la operación realizada por el procedimiento almacenado
      Repeat
         DELAY PROCESS(Current process;300)
         GET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
         If(Undefined(spErrCode))
  ` Nota: si el procedimiento almacenado no ha inicializado su propia instancia
  ` de la variable spErrCode, puede que retorne una variable indefinida
            spErrCode:=1
         End if
      Until(spErrCode<=0)
  ` Enviemos un acuse de recibo al procedimiento almacenado
      spErrCode:=1
      SET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
      $vhHoraFinal:=Current time
      ALERT("La operación tardó"+String(0+($vhHoraFinal-$vhHoraInicial))+" segundos.")
   Else
      ALERT("No hay suficiente memoria para cargar el documento.")
   End if
End if
```

*He aquí el método de proyecto **SERVIDOR IMPORTAR** ejecutado como procedimiento almacenado:*

```
  ` Método de proyecto SERVIDOR IMPORTAR
  ` SERVIDOR IMPORTAR ( Entero largo ; Alfa ; BLOB )
```

```
  ` SERVIDOR IMPORTAR ( Número de tabla ; Formulario de entrada ; Datos importados )

C_LONGINT($1)
C_TEXT($2)
C_BLOB($3)
C_LONGINT(spErrCode)

  ` La operación no ha terminado aún, asignemos 1 a spErrCode
spErrCode:=1
$vpTabla:=Table($1)
FORM SET INPUT($vpTabla->;$2)
$vsDocNombre:="Archivo import "+String(1+Random)
DELETE DOCUMENT($vsDocNombre)
BLOB TO DOCUMENT($vsDocNombre;$3)
IMPORT TEXT($vpTabla->;$vsDocNombre)
DELETE DOCUMENT($vsDocNombre)
  ` La operación ha terminado, asignemos 0 a spErrCode 0
spErrCode:=0
  ` Espere a que el equipo cliente reciba los resultados
Repeat
   DELAY PROCESS(Current process;1)
Until(spErrCode>0)
```

Una vez estos dos métodos de proyecto hayan sido implementados en su base de datos, puede efectuar una importación basada en un procedimiento almacenado, escribiendo por ejemplo:

```
CLIENTE IMPORTAR(->[Tabla1];"Import")
```

Con algunas pruebas comparativas, puede comprobar que con este tipo de método, la importación de los registros puede ser 60 veces más rápida que una importación clásica.

Get process activity

Get process activity {(opciones)} -> Resultado

Parámetro	Tipo		Descripción
opciones	Entero largo	➡	Opciones a devolver
Resultado	Objeto	⮌	Instantánea de los procesos en ejecución y/o (4D Server únicamente) de las sesiones de usuario

Descripción

*El comando **Get process activity** devuelve una instantánea de las sesiones de los usuarios conectados y/o de los procesos relacionados que se ejecutan en un momento dado. Este comando devuelve todos los procesos, incluyendo los procesos internos que no eran accesibles por el comando **PROCESS PROPERTIES**.*

- *Cuando se ejecuta en el servidor, por defecto si omite el parámetro opciones, **Get process activity** devuelve las listas de sesiones usuario y de los procesos, como se muestra a continuación:*

```
{
"sessions": [
            {
                "type": "remote",
                "userName": "Designer",
                "machineName": "iMac27caroline",
                "systemUserName": "Caroline Briaud",
                "IPAddress": "192.168.18.18",
                "hostType": "mac",
                "creationDateTime": "2017-09-22T12:46:39Z",
                "state": "postponed",
                "ID": "3C81A8D7AFE64C2E9CCFFCDC35DC52F5"
             },...
        ],
"processes": [
            {
                "name": "Application process",
                "sessionID": "3C81A8D7AFE64C2E9CCFFCDC35DC52F5",
                "number": 4,
                "ID": 4,
                "visible": true,
                "systemID": "123145476132864",
                "type": -18,
                "state": 0,
                "cpuUsage": 0,
                "cpuTime": 0.006769,
                "preemptive": false,
                "session": {
                            "type": "remote",
                            "userName": "Designer",
                            "machineName": "iMac27caroline",
                            "systemUserName": "Caroline Briaud",
                            "IPAddress": "192.168.18.18",
                            "hostType": "mac",
                            "creationDateTime": "2017-09-22T12:46:39Z",
                            "state": "postponed",
                            "ID": "3C81A8D7AFE64C2E9CCFFCDC35DC52F5"
                             }
            },...
    ]
}
```

*Se puede seleccionar la lista a devolver al pasar una de las siguientes constantes del tema "**Entorno 4D**" en el parámetro opciones:*

Constante	Tipo	Valor	Comentario
Processes only	Entero largo	1	Devuelve sólo la lista de procesos
Sessions only	Entero largo	2	Devuelve sólo la lista de sesión de usuario

- *Cuando se ejecuta en 4D en modo remoto o local, **Get process activity** sólo devuelve la lista de procesos (el parámetro opciones es inútil).*

Lista de sesiones

La propiedad "sessions" contiene una colección de objetos que describen todas las sesiones usuario en ejecución en el servidor, por ejemplo:

```
{
    "sessions": [
        {
            "type": "remote",
            "hostType": "mac",
            "userName": "Designer",
            "machineName": "YANNICK-CORE-I7",
            "systemUserName": "Yannick",
            "IPAddress": "fe80::c189:5ea1:c635:90e6",
            "creationDateTime": "2017-04-24T17:13:15Z",
            "state": "postponed",
            "ID": "63DCD177059C60459F2AD278EAD84B4B"
        },
        ...
    ],...
}
```

Cada objeto sesión contiene las propiedades siguientes (cuando no están disponibles, las propiedades no se devuelven):

Nombre de la propiedad	***Tipo***	***Descripción***
type	*Texto (enum)*	*Tipo de sesión. Valores posibles: "remote", "storedProcedure", "web", "rest"*
hostType	*Texto (enum)*	*Tipo de host. Valores posibles: "windows", "mac", "browser"*
userName	*Texto*	*Nombre de usuario*
machineName	*Texto*	*Nombre de la máquina remota*
systemUserName	*Texto*	*Nombre de la sesión del sistema abierta en la máquina remota*
IPAddress	*Texto*	*Dirección IP de la máquina remota*
creationDateTime	*Date ISO 8601*	*Fecha y hora de la conexión del equipo remoto*
state	*Texto (enum)*	*Estado de las sesión. Valores posibles: "active", "postponed", "sleeping"*
ID	*Texto*	*UUID de la sesión*

Process list

La propiedad "proceso" contiene una colección de objetos que describen todos los procesos en ejecución en el servidor, por ejemplo:

```
{
    "processes": [
        {
            "state": 0,
            "cpuTime": 12.755332415303,
            "preemptive": true
        },
        {
            "name": "processOnServer",
            "sessionID": "BB8A0C8420FE5741B59F3B13F40A3063",
            "number": 6,
            "ID": 6,
            "visible": true,
            "type": 1,
            "state": 0,
            "cpuTime": 0.40872755886655,
            "preemptive": false
        },
        {
            "name": "ServerNet select I/O handler",
            "state": 0,
            "cpuTime": 0.015625,
            "preemptive": true
        },...
    ]
}
```

Cada objeto proceso contiene las siguientes propiedades (cuando no están disponibles, las propiedades no se devuelven):

Nombre	Tipo	Descripción
name	Texto	Nombre del proceso
sessionID	Texto	UUID de la sesión
type	Entero largo	Tipo de proceso en ejecución. Puede utilizar las siguientes constantes del tema "**Tipo de proceso**":

Constante	Valor	Comentario
HTTP Log flusher	-58	
Logger process	-57	
HTTP Listener	-56	
HTTP Worker pool server	-55	
SQL Listener	-54	
SQL Net Session manager	-53	
SQL Worker pool server	-52	
DB4D Listener	-51	
DB4D Mirror	-50	
DB4D Cron	-49	
DB4D Worker pool user	-48	
DB4D Garbage collector	-47	
DB4D Flush cache	-46	
DB4D Index builder	-45	
ServerNet Session manager	-44	
ServerNet Listener	-43	
Worker pool spare	-42	
Worker pool in use	-41	
Other internal process	-40	
Main 4D process	-39	
Client manager process	-31	
Compiler process	-29	
Monitor process	-26	
Internal timer process	-25	
SQL Method execution process	-24	
MSC process	-22	
Restore Process	-21	
Log file process	-20	
Backup process	-19	
Internal 4D server process	-18	
Method editor macro process	-17	
On exit process	-16	
Server interface process	-15	
Execute on client process	-14	
Web server process	-13	
Web process on 4D remote	-12	
Other 4D process	-10	
External task	-9	
Event manager	-8	
Apple event manager	-7	
Serial Port Manager	-6	
Indexing process	-5	
Cache manager	-4	
Web process with no context	-3	
Design process	-2	
Main process	-1	
None	0	
Execute on server process	1	
Created from menu command	2	
Created from execution dialog	3	
Other user process	4	
Worker process	5	Procesos Worker lanzado por el usuario

state	Entero largo	Estado actual (ver la lista de constantes **Estado del proceso**)
cpuTime	Real	Tiempo de ejecución (segundos)
cpuUsage	Real	Porcentaje de tiempo dedicado a este proceso (entre 0 y 1)
preemptive	Booleano	si ejecuta apropiativo,de lo contrario false
visible	Booleano	True si es visible, de lo contrario false
ID	Entero largo	Identificador único del proceso
number	Entero largo	Número del proceso

Ejemplo

Usted quiere obtener la colección de todas las sesiones de usuario:

```
 //A ser ejecutado en el servidor
C_OBJECT($result)
C_COLLECTION($userCol)
$result:=Get process activity(Sessions only)
$userCol:=OB Get($result;"users")
```

GET REGISTERED CLIENTS

GET REGISTERED CLIENTS (listaClientes ; metodos)

Parámetro	Tipo		Descripción
listaClientes	Array texto	⇐	Lista de 4D Clients registrados
metodos	Array entero largo	⇐	Lista de los métodos a ejecutar

Descripción

El comando ***GET REGISTERED CLIENTS*** *llena dos arrays:*

- *listaClients contiene la lista de los clientes "registrados" utilizando el comando* ***REGISTER CLIENT****.*
- *metodos proporciona la lista de "cargas de trabajo" de cada cliente. La carga de trabajo es el número de métodos que un cliente 4D debe ejecutar llamando el comando* ***EXECUTE ON CLIENT*** *(para mayor información, por favor consulte la descripción del comando* ***EXECUTE ON CLIENT****).*

Nota: *si la operación fue exitosa, la variable sistema* ***OK*** *es igual a 1.*

Ejemplo 1

Asumamos que quiere obtener una lista de todos los clientes registrados y los métodos que falta ejecutar:

```
ARRAY TEXT($clientes;0)
ARRAY LONGINT($metodos;0)
GET REGISTERED CLIENTS($clientes;$metodos)
```

Ejemplo 2

Consulte el ejemplo del comando ***REGISTER CLIENT****.*

Variables y conjuntos del sistema

Si la operación se realiza correctemente, la variable sistema OK toma el valor 1.

New process

New process (metodo ; pila {; nombre {; param {; param2 ; ... ; paramN}}}{; *}) -> Resultado

Parámetro	Tipo		Descripción
metodo	Cadena	➡	Método a ejecutar en el proceso
pila	Entero largo	➡	Tamaño de la pila en bytes
nombre	Cadena	➡	Nombre del proceso creado
param	Expresión	➡	Parámetros del método
*	Operador	➡	Proceso único
Resultado	Entero largo	↩	Número del proceso creado recientemente o del proceso que está siendo ejecutado

Descripción

*El comando **New process** inicia un nuevo proceso (en el mismo equipo) y devuelve el número de este proceso.*

*Si no se pudo crear el proceso (por ejemplo, si no hay suficiente memoria), **New process** devuelve cero (0) y se genera un error. Puede interceptar este error utilizando un método de gestión de errores instalado por el comando **ON ERR CALL**.*

Método de proceso

En metodo, se pasa el nombre del método de gestión del nuevo proceso. Una vez que 4D haya definido el contexto para el nuevo proceso, comienza la ejecución de este método que se convierte en el método de proceso.

Pila del proceso

El parámetro pila, le permite indicar la cantidad de memoria asignada para la pila del proceso. Este valor representa el espacio en memoria utilizado para "apilar" las llamadas de métodos, las variables locales, los parámetros de subrutinas y los registros apilados.

- *Pase 0 en pila para utilizar un tamaño de pila predeterminado, adecuado para la mayoría de aplicaciones (valor recomendado).*
- *En ciertos casos particulares, es posible que desee utilizar un valor personalizado. Debe expresarse en bytes. Se recomienda pasar un mínimo de 64 KB (alrededor de 64.000 bytes) y puede utilizar valores sobre 512 KB, en particular si el proceso puede realizar llamadas a cadenas largas (subrutinas llamando subrutinas en cascada).*

***Nota:** la pila NO es la memoria total reservada para el proceso. Los procesos comparten memoria para los registros, las variables interproceso, etc. Un proceso utiliza igualmente la memoria extra para almacenar sus variables proceso. La pila contiene diferente información 4D: la cantidad de información depende del número de llamadas de métodos anidados, el número de formularios que abrirá antes de cerrarlos y el número y tamaño de variables locales utilizadas en cada llamada de método anidado.*

***Nota versiones 64 bits**: la pila de un proceso ejecutado en una versión 64 bits de 4D requiere generalmente una cantidad de memoria más importante que en 4D 32 bits (alrededor del doble). Asegúrese de verificar este parámetro cuando su código esté destinado a ser ejecutado en una versión 64 bits de 4D.*

Nombre del proceso

*Pase el nombre del nuevo proceso en nombre. Este nombre aparecerá en la lista de procesos del Explorador de ejecución y será devuelto por el comando **PROCESS PROPERTIES** cuando se aplica a este nuevo proceso. Puede omitir este parámetro; si lo hace, el nombre del proceso será una cadena vacía. Puede crear un proceso local colocando como prefijo el símbolo dólar ($).*

***Importante:** Recuerde que en cliente/servidor, los procesos locales no deben acceder a los datos.*

Parámetros del método proceso

*Puede pasar parámetros al método proceso utilizando uno o más parámetros param. Puede pasar parámetros de la misma manera que para las subrutinas (ver la sección **Pasar parámetros a los métodos**). Una vez que haya comenzado la ejecución en el contexto del nuevo proceso, el método proceso recibe los valores de los parámetros en $1, $2 , etc. Recuerde que los arrays no pueden pasarse como parámetros a un método. Además, estas consideraciones adicionales deben ser tomadas en cuenta en el contexto del comando **New process**:*

- *Se permite punteros a tablas o campos.*
- *Los punteros a las variables, particularmente las variables locales y proceso, no se recomiendan ya que estas variables pueden estar indefinidas en el momento en que método proceso las accede.*
- *Si pasa un parámetro de tipo Objeto o una Colección, 4D creará en este caso una copia del objeto en el proceso de destino en lugar de una referencia.*

***Nota**: si pasa los parámetros al método proceso, debe pasar el parámetro nombre; no puede omitirse en este caso.*

Parámetro opcional *

Le indica a 4D que debe verificar primero si se está ejecutando un proceso con el nombre que usted pasó en nombre. Si es así, 4D no inicia un nuevo proceso y devuelve el número del proceso con ese nombre.

Ejemplo

Dado el siguiente método proyecto:

```
` ADD CUSTOMERS
SET MENU BAR(1)
Repeat
```

```
    ADD RECORD([Customers];*)
Until(OK=0)
```

*Si asocia este método de proyecto a un comando de menú creado en el Editor de barras de menú y le asigna la propiedad Iniciar un nuevo proceso, 4D automáticamente iniciará un nuevo proceso en el momento de la ejecución del método. La llamada **SET MENU BAR**(1) asocia esta barra de menús al nuevo proceso. En ausencia de ventanas (que podría haber abierto con **Open window**), la llamada a **ADD RECORD** abrirá automáticamente una.*

Para poder iniciar el proceso Add Customers haciendo clic en un botón situado en un panel de control personalizado, puede escribir:

```
  ` Método de objeto botón bAddCustomers
$vlProcessID:=New process("Add Customers";0;"Adding Customers")
```

El botón hace lo mismo que el comando de menú personalizado.

*Si cuando selecciona el elemento de menú o hace clic en el botón, usted quiere que el proceso comience (si no existe) o pasado al primer plano (si ya se está ejecutando), puede crear el método **START ADD CUSTOMERS**:*

```
  ` START ADD CUSTOMERS
$vlProcessID:=New process("Add Customers";0;"Adding Customers";*)
If($vlProcessID#0)
    BRING TO FRONT($vlProcessID)
End if
```

El método de objeto de bAddCustomers se convierte en:

```
  ` Método objeto del botón bAddCustomers
START ADD CUSTOMERS
```

*En el editor de barras de menús, puede reemplazar **ADD CUSTOMERS** por el método **START ADD CUSTOMERS**, y deseleccionar la propiedad **Iniciar un nuevo proceso** para el comando de menú.*

PAUSE PROCESS

PAUSE PROCESS (proceso)

Parámetro	***Tipo***		***Descripción***
proceso	*Entero largo*	➡	*Número de proceso*

Descripción

PAUSE PROCESS *suspende la ejecución de proceso hasta que es reactivada por el comando* ***RESUME PROCESS****. Durante este periodo, process no toma tiempo en su equipo. Aunque un proceso esté suspendido, el proceso aún está en memoria.*

Si process ya está suspendido, ***PAUSE PROCESS*** *no hace nada. Si el proceso ha sido retrasado utilizando el comando* ***DELAY PROCESS****, se suspende el proceso. Si recibe la orden* ***RESUME PROCESS*** *retoma el proceso inmediatamente.*

Cuando se suspende la ejecución de un proceso, las ventanas que pertenecen a este proceso no son editables. En este caso, para evitar confundir al usuario, considere ocultar el proceso. Si process no existe, el comando no hace nada.

Advertencia: *utilice* ***PAUSE PROCESS*** *únicamente con procesos que usted ha creado.* ***PAUSE PROCESS*** *no tiene ningún efecto en el proceso principal.*

Nota: *no puede utilizar este comando para asignar un procedimiento almacenado en el equipo servidor de un equipo cliente. (proceso<0).*

Process aborted

Process aborted -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		True = el proceso va a abortarse, False = el proceso no va a ser abortado

Descripción

*El comando **Process aborted** devuelve **True** si el proceso en el cual se llama está a punto de ser interrumpido inesperadamente, es decir que la ejecución del comando no pudo terminar se ejecución "normal". Por ejemplo, esto puede ocurrir después de llamar **QUIT 4D**.*

Process number

Process number (nombre {; *}) -> Resultado

Parámetro	Tipo		Descripción
nombre	Cadena	→	Nombre del proceso del cual recuperar el número
*		→	Devolver el número del proceso servidor
Resultado	Entero largo	←	Número de proceso

Descripción

Process number *devuelve el número del proceso cuyo nombre se pasa en nom. Si no se encuentra ningún proceso,* ***Process number*** *devuelve 0.*

*El parámetro opcional * le permite recuperar, a partir de 4D Client, el número de un proceso que se está ejecutando en el servidor (un procedimiento almacenado). En este caso, el valor devuelto es negativo. Esta opción es particularmente útil cuando se utilizan los comandos* ***GET PROCESS VARIABLE*** *y* ***SET PROCESS VARIABLE****. Para mayor información, consulte las descripciones de estos comandos.*

*Si el comando se ejecuta con el parámetro * desde un proceso en el equipo servidor, el valor devuelto es positivo.*

Ejemplo

Usted crea una paleta flotante, que corre en un proceso separado, en el cual usted implementa sus propias herramientas para interactuar con el entorno Diseño. Por ejemplo, cuando selecciona un elemento en una lista jerárquica de palabras claves, usted quiere pegar algún texto en la ventana del primer plano del entorno Diseño. Para hacerlo, puede utilizar el Portapapeles, pero el evento de pegado debe ocurrir dentro del proceso Diseño. La siguiente función devuelve el número del proceso del proceso Diseño (si está activo):

```
` Método de proyecto Numero proceso Diseño
` Numero proceso Diseño -> Entero largo
` Numero proceso Diseño -> Número del proceso de Diseño

$0:=Process number("Proceso Diseño")
` Nota: Esto puede no funcionar en el futuro si la fuente cambia
```

Con esta función, el método de proyecto listado pega el texto recibido como parámetro en la ventana del primer plano en el entorno Diseño (si aplica):

```
` Método de proyecto PEGAR TEXTO EN DISEÑO
` PEGAR TEXTO EN DISEÑO ( Text )
`PEGAR TEXTO EN DISEÑO (Texto a pegar en la ventana de Diseño del primer plano)

C_TEXT($1)
C_LONGINT($vlDiseñoPID;$vlCont)

$vlDiseñoPID:=Numero proceso Diseño
If($vlDiseñoPID #0)
  ` Coloque el texto en el portapapeles
  SET TEXT TO PASTEBOARD($1)
  ` Generar un evento Ctrl-V / Comando-V
  POST KEY(ASCII("v");Command key mask;$vlDiseñoPID)
  ` Llamar repetitivamente DELAY PROCESS para que el minutero puede pasar
  ` sobre el evento al proceso Diseño
  For($vlCont;1;5)
    DELAY PROCESS(Current process;1)
  End for
End if
```

PROCESS PROPERTIES

PROCESS PROPERTIES (proceso ; procNom ; procEstado ; procTiempo {; procVisible {; unicoID {; origen}}})

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número del proceso
procNom	Cadena	⬅	Nombre del proceso
procEstado	Entero largo	⬅	Estado del proceso
procTiempo	Entero largo	⬅	Tiempo acumulado de ejecución del proceso en tics
procVisible	Booleano, Entero largo	⬅	Visible (TRUE) u Oculto (FALSE)
unicoID	Entero largo	⬅	Número único del proceso
origen	Entero largo	⬅	Origen del proceso

Descripción

*El comando **PROCESS PROPERTIES** devuelve diferente información sobre el proceso cuyo número de proceso se pasa en proceso.*

Después de la llamada:

- *procNom devuelve el nombre del proceso. Algunos puntos a tener en cuenta acerca del nombre del proceso:*
  - *Si el proceso fue iniciado desde la caja de diálogo **Ejecutar un método** (con la opción **Nuevo proceso** seleccionada), su nombre es "**P_**" seguido por un número.*
  - *Si el proceso fue iniciado a partir de un comando de menú personalizado cuya propiedad **Iniciar un nuevo proceso** es seleccionada, el nombre del proceso es "**M_**" o "**ML_**" seguido por un número.*
  - *Si el proceso fue iniciado por el servidor web, su nombres es "Web Process#" seguido por un UUID.*
  - *Si el proceso ha sido suspendido (y su "espacio" no ha sido reutilizado), aún se devuelve el nombre del proceso. Para detectar si un proceso esta suspendido, pruebe procEstado=-1 (ver a continuación).*
- *procEstado devuelve el estado del proceso en el momento de la llamada. Este parámetro puede devolver uno de los valores ofrecidos por las siguientes constantes predefinidas:*

Constante	Tipo	Valor
Does not exist	Entero largo	-100
Aborted	Entero largo	-1
Executing	Entero largo	0
Delayed	Entero largo	1
Waiting for user event	Entero largo	2
Waiting for input output	Entero largo	3
Waiting for internal flag	Entero largo	4
Paused	Entero largo	5
Hidden modal dialog	Entero largo	6

- *procTiempo devuelve el tiempo acumulado que el proceso ha utilizado desde que comenzó, en tics (1/60 de segundo).*
- *El parámetro opcional procModo puede ser una variable de tipo booleano o entero largo:*
- *Si es de tipo Booleano, devuelve True si el proceso es visible y False si está oculto.*
- *Si es de tipo entero largo, después de la ejecución del método, contiene un campo de bits donde los dos primeros bits son definidos:*
  - *bit 0 devuelve la propiedad de visibilidad: 1 si el proceso es visible y 0 si está oculto*
  - *bit 1 devuelve la propiedad de modo apropiativo: 1 si el proceso se ejecuta en modo apropiativo y 0 si se ejecuta en modo cooperativo.*
    ***Nota:** esta propiedad sólo es útil en aplicaciones de 64 bits, donde los procesos pueden ejecutarse en modo apropiativo o cooperativo. Para mayor información, consulte la sección **Procesos 4D apropiativos**.*
- *unicoID, si se especifica, devuelve el número único del proceso. De hecho, cada proceso tiene un número de proceso así como un número único de proceso por sesión. Éste último permite diferenciar entre dos procesos o sesiones de proceso. Corresponde al número de procesos que han sido iniciados durante la sesión de la aplicación 4D.*
- *origen, si se especifica, devuelve un valor que describe el origen del proceso. Este parámetro puede devolver uno de los valores de las siguientes constantes predefinidas (en el tema "**Tipo de proceso**"):*
  [#table_kst them="4727"
  remove="861928,3423082,3423089,3423075,3423068,3423061,3423054,3423047,3423039,3423025,3423018,3423010,342

***Nota:** los procesos internos de 4D devuelven un valor negativo y los procesos generados por el usuario devuelven un valor positivo.*

*Si el proceso no existe, significa que no pasó un número incluido en el intervalo de 1 a **Count tasks**, **PROCESS PROPERTIES** deja sin modificar los valores de las variables pasados en parámetros.*

Ejemplo 1

El siguiente ejemplo devuelve el nombre, el estado, el tiempo tomado en las variables vNom, vEstado, y vTiempoTransc para el proceso actual:

```
C_TEXT(vNom)  ` Inicializar las variables
C_LONGINT(vEstado)
```

```
C_LONGINT(vTiempoTransc)
PROCESS PROPERTIES(Current process;vNom;vEstado;vTiempoTransc)
```

Ejemplo 2

*Ver el ejemplo de la sección **Método de base On Exit Database**.*

Ejemplo 3

Usted quiere conocer la visibilidad y el modo de ejecución del proceso actual. Puede escribir:

```
C_TEXT(vName)
C_LONGINT(vState)
C_LONGINT(vTime)
C_LONGINT(vFlags)
C_BOOLEAN(isVisible)
C_BOOLEAN(isPreemptive)
PROCESS PROPERTIES(Current process;vName;vState;vTime;vFlags)
isVisible:=vFlags?? 0 //true si visible
isPreemptive:=vFlags?? 1 //true si apropiativo
```

Process state

Process state (proceso) -> Resultado

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número de proceso
Resultado	Entero largo	⮌	Estado del proceso

Descripción

*El comando **Process state** devuelve el estado del proceso cuyo número se pasó en proceso.*
El resultado de la función puede ser uno de los valores de las siguientes constantes predefinidas:

Constante	Tipo	Valor
Does not exist	Entero largo	-100
Aborted	Entero largo	-1
Executing	Entero largo	0
Delayed	Entero largo	1
Waiting for user event	Entero largo	2
Waiting for input output	Entero largo	3
Waiting for internal flag	Entero largo	4
Paused	Entero largo	5
Hidden modal dialog	Entero largo	6

*Si el proceso no existe (lo cual significa que no pasó un número en el rango de 1 a **Count tasks**), **Process state** devuelve **Does not exist** (-100).*

Ejemplo

El siguiente ejemplo coloca el nombre y número de referencia para cada proceso en los arrays asProcName y aiProcNum. El método prueba si el proceso ha sido abortado. En este caso, el nombre y el número del proceso no son añadidos a los arrays:

```
$vlNbTareas:=Count tasks
ARRAY TEXT(asProcNombre;$vlNbTareas)
ARRAY INTEGER(aiProcNum;$vlNbTareas)
$vlActualCont:=0
For($vlProcess;1;$vlNbTareas)
   If(Process state($vlProcess)>=Executing)
      $vlActualCont:=$vlActualCont+1
      PROCESS PROPERTIES($vlProcess;asProcNombre{$vlActualCont};$vlStado;$vlHora)
      aiProcNum{$vlActualCont}:=$vlProcess
   End if
End for
  ` Eliminar los elementos extras superfluos
ARRAY TEXT(asProcNombre;$vlActualCont)
ARRAY INTEGER(aiProcNum;$vlActualCont)
```

REGISTER CLIENT

REGISTER CLIENT (nomCliente {; periodo}{; *})

Parámetro	Tipo		Descripción
nomCliente	Cadena	➡	Nombre de la sesión 4D Client
periodo	Entero largo	➡	**Ignorado desde la versión 11.3***
*	Operador	➡	Proceso local

Descripción

*El comando **REGISTER CLIENT** "registra" un equipo cliente 4D con el nombre especificado en nomCliente en 4D Server, con el fin de permitir a otros clientes o eventualmente 4D Server (utilizando procedimientos almacenados) ejecutar métodos utilizando el comando **EXECUTE ON CLIENT**. Una vez registrado, un cliente 4D puede ejecutar uno o varios métodos para otros clientes.*

***Notas**:*

- *También puede registrar automáticamente cada puesto cliente que se conecte a 4D Server utilizando la opción "Registrar los clientes al inicio..." en la caja de diálogo de Preferencias.*
- *Si este comando se utiliza con 4D en modo local, no tiene efecto.*
- *Más de una estación 4D client puede tener el mismo nombre registrado.*

*Cuando se ejecuta este comando, un proceso, llamado nomCllient, se crea en el equipo cliente. Este proceso sólo puede ser abortado por el comando **UNREGISTER CLIENT**.*
*Si pasa el parámetro opcional *, el proceso creado es local. 4D añade automáticamente el signo dólar ($) al comienzo del nombre del proceso. De lo contrario, el proceso es global.*

***Nota de compatibilidad:** a partir de la versión 11.3 de 4D, se han optimizado los mecanismos de comunicación servidor/cliente. Ahora el servidor envía las peticiones de ejecución directamente a los clientes registrados cuando es necesario (tecnología "push"). El principio anterior donde los clientes buscaban periodicamente el servidor, ya no se usa. El parámetro periodo se ignora si se pasa.*

*Una vez ejecutado el comando, no es posible modificar rápidamente el nombre del cliente 4D o el periodo de interrogación al servidor. Para hacerlo, debe llamar al comando **UNREGISTER CLIENT**, y luego ejecutar el comando **REGISTER CLIENT**.*

*Si un cliente 4D está registrado correctamente, la variable sistema **OK** es igual a 1. Si el cliente 4D ya fue registrado, el comando no hace nada y **OK** toma el valor 0.*

Ejemplo

En el siguiente ejemplo, vamos a crear un sistema de mensajería pequeño que permita a los puestos clientes comunicarse entre ellos.

1) Este método, Registro, le permite registrar un cliente 4D y conservarlo listo para recibir un mensaje de otro 4D Client:

```
   `Debe salir del registro antes de registrarse con otro nombre
UNREGISTER CLIENT
Repeat
   vNomPseudo:=Request("Introduzca su nombre:";"Usuario";"OK";"Cancelar")
Until((OK=0)|(vNomPseudo#""))
If(OK=0)
   ... ` No hacer nada
Else
   REGISTER CLIENT(vNomPseudo)
End if
```

*2) La siguiente instrucción le permite obtener una lista de los 4D Clients registrados. Puede colocarse en el **Método de base On Startup**:*

```
PrListClient:=New process("Lista_4D Client";32000;"Lista de clientes registrados")
```

3) El método Lista_4DClient le permite recuperar todos los 4D Clients registrados y las personas que pueden recibir mensajes:

```
If(Application type=4D Client)
  ` el código a continuación sólo es válido en modo cliente-servidor
   $Ref:=Open window(100;100;300;400;-(Palette window+Has window title);"Lista de clientes registrados")
   Repeat
      GET REGISTERED CLIENTS($ListClient;$ListeCharge)
  `Retrieve the registered clients in $ClientList
      ERASE WINDOW($Ref)
      GOTO XY(0;0)
      For($p;1;Size of array($ListClient))
         MESSAGE($ListClient{$p}+Char(Carriage return))
      End for
  `Mostrar cada segundo
```

```
        DELAY PROCESS(Current process;60)
    Until(False) ` Bucle infinito
End if
```

4) El siguiente método le permite enviar un mensaje a otro cliente 4D registrado. Llama al método Mostrar_Mensaje (ver a continuación).

```
$Destinatario:=Request("Destinatario del mensaje:";"")
  ` Introduza el nombre de las personas visibles en la ventana generada por el
  ` Método de base On Startup
If(OK#0)
    $Mensaje:=Request("Mensaje:") ` mensaje
    If(OK#0)
        EXECUTE ON CLIENT($Destinatario;"Mostrar_Mensaje";$Mensaje) ` Enviar mensaje
    End if
End if
```

5) Este es el método Mostrar_Mensaje:

```
C_TEXT($1)
ALERT($1)
```

6) Finalmente, este método permite a un puesto cliente no ser visible para los otros clientes 4D y no recibir más mensajes:

```
UNREGISTER CLIENT
```

Variables y conjuntos del sistema

*Si el cliente 4D está registrado correctamente, la variable sistema **OK** es igual a 1. Si 4D Client ya fue registrada, el comando no hace nada y **OK** toma el valor 0.*

RESUME PROCESS

RESUME PROCESS (proceso)

Parámetro	***Tipo***		***Descripción***
proceso	*Entero largo*	➡	*Número de proceso*

Descripción

RESUME PROCESS *reactiva un proceso cuya ejecución ha sido retrasada o suspendida. Si proceso no está retrasado o suspendido,* ***RESUME PROCESS*** *no hace nada.*

Si proceso ha sido retrasado anteriormente, consulte los comandos ***PAUSE PROCESS*** *o* ***DELAY PROCESS****. Si proceso no existe, el comando no hace nada.*

Nota: *no puede utilizar este comando para asignar un procedimiento almacenado en el equipo servidor de un equipo cliente. (proceso<0).*

UNREGISTER CLIENT

UNREGISTER CLIENT
Este comando no requiere parámetros

Descripción

*El comando **UNREGISTER CLIENT** da de baja un registro de un equipo 4D Client. El cliente debe haber sido registrado por el comando **REGISTER CLIENT.***
***Nota:** un cliente 4D se saca del registro automáticamente cuando el usuario sale de la aplicación.*
Si el equipo cliente no fue registrado previamente o si el comando se ejecutó en 4D en modo local, el comando no hace nada.
*Si el cliente salió del registro correctamente, la variable sistema **OK** es igual a 1. Si el cliente no fue registrado, **OK** es igual a 0.*

Ejemplo

*Consulte el ejemplo del comando **REGISTER CLIENT**.*

Variables y conjuntos del sistema

Si el cliente es dado de baja correctamente, la variable sistema OK toma el valor 1, de lo contrario toma el valor 0.

Procesos (Comunicación)

- *Sobre los semáforos*
- *Sobre workers*
- *CALL WORKER*
- *CLEAR SEMAPHORE*
- *GET PROCESS VARIABLE*
- *KILL WORKER*
- *Semaphore*
- *SET PROCESS VARIABLE*
- *Test semaphore*
- *VARIABLE TO VARIABLE*

Sobre los semáforos

¿Qué es un semáforo?

En un programa de ordenador, un semáforo es una herramienta que se utiliza para protegerse de las acciones que deben ser realizadas por un único proceso o usuario a la vez.

En 4D, la necesidad convencional de uso de un semáforo es para modificar un array interproceso: si un proceso está modificando los valores del array, otro proceso no debe poder hacer lo mismo al mismo tiempo. El desarrollador utiliza un semáforo para indicar a un proceso que sólo puede realizar su secuencia de operaciones si ningún otro proceso está llevando a cabo las mismas tareas. Cuando un proceso se encuentra con un semáforo, hay tres posibilidades:

- *Obtiene inmediatamente el derecho a pasar*
- *Espera su turno hasta que obtiene el derecho a pasar*
- *Continúa su camino, abandonando la idea de realizar las tareas.*

Por lo tanto, el semáforo protege partes del código. Se permite pasar sólo un proceso a la vez y bloquea el acceso hasta que el proceso que tiene actualmente el derecho de uso renuncia a este derecho liberando el semáforo.

Comandos de manipulación de los semáforos

*En 4D, se establece un semáforo llamando a la función **Semaphore**. Para liberar un semáforo, se llama al comando **CLEAR SEMAPHORE**.*

*La función **Semaphore** tiene un comportamiento muy especial ya que realiza potencialmente dos acciones a la vez:*

- *Si el semáforo ya está asignado, la función devuelve **True***
- *Si no se asigna el semáforo, la función lo asigna al proceso y devuelve **False** al mismo tiempo.*

Esta doble acción realizada por el mismo comando asegura que ninguna operación externa se puede insertar entre la prueba del semáforo y su asignación.

*Puede utilizar el comando **Test semaphore** para saber si un semáforo ya está asignado o no. Este comando se utiliza principalmente como parte de las operaciones largas, tales como el cierre anual de las cuentas, en donde **Test semaphore** le permite controlar la interfaz para evitar el acceso a ciertas operaciones tales como la adición de los datos contables.*

¿Cómo utilizar los semáforos?

Los semáforos deben utilizarse respetando los siguientes principios:

- *un semáforo se debe definir y lanzar en el mismo método,*
- *la ejecución de código protegido por el semáforo debe ser lo más corta posible,*
- *el código debe ser temporizado por medio del parámetro contTics de la función **Semaphore** para esperar la liberación del semáforo.*

Este es el código típico para el uso de un semáforo:

```
While(Semaphore("MySemaphore";300))
   IDLE
End while
 // ubicar aquí el código protegido por el semáforo
CLEAR SEMAPHORE("MySemaphore")
```

Un semáforo que no se libera puede bloquear parte de la base. Configurar y liberar el semáforo en el mismo método ayuda a eliminar este riesgo.

Minimizar el código protegido por el semáforo aumenta la fluidez de la aplicación y evita que el semáforo sea un cuello de botella.

*Por último, utilizando el parámetro opcional contTics del comando **Semaphore** es esencial para optimizar la espera del semáforo a liberar. Utilizando este parámetro, los comandos funcionan de la siguiente manera:*

- *El proceso espera un máximo del número especificado de tics (300 en el ejemplo) para que el semáforo esté disponible, sin la ejecución de código pasa a la siguiente línea,*
- *Si se libera el semáforo antes del final de este límite, se le asigna inmediatamente al proceso (**Semaphore** devuelve False) y se reanuda la ejecución de código,*
- *Si el semáforo no se libera antes del final de este límite y se reanuda la ejecución del código.*

El comando también da prioridad a las peticiones estableciendo una cola. De esta manera, el primer proceso que solicita un semáforo será el primero en obtener uno.

Tenga en cuenta que el tiempo de espera se establece en función de las características específicas de la aplicación.

Semáforo local o global

Hay dos tipos de semáforos en 4D: semáforos locales y semáforos globales.

- *Un semáforo local es visible para todos los procesos de un mismo puesto y sólo en el puesto. Un semáforo local puede ser creado al añadir un prefijo al nombre del semáforo un signo de dólar ($). Se utiliza semáforos locales para supervisar las*

operaciones entre los diferentes procesos que se ejecutan en el misma equipo. Por ejemplo, un semáforo local se puede utilizar para controlar el acceso a un array interproceso compartido por todos los procesos de una base de datos mono usuario o de un equipo cliente.

- *Un semáforo global es accesible a todos los usuarios y todos sus procesos. Los semáforos globales se utilizan para controlar las operaciones entre los usuarios de una base de datos multiusuario.*

Los semáforos globales y locales son idénticos en su lógica. La diferencia reside en su alcance.

En el modo cliente-servidor, los semáforos globales se comparten entre todos los procesos que se ejecutan en todos los clientes y servidores. Un semáforo local solamente se comparte entre los procesos que se ejecutan en la máquina donde se ha creado.

En 4D, los semáforos globales o locales tienen el mismo alcance, ya que usted es el único usuario. Sin embargo, si su base se está utilizando en ambas configuraciones, asegúrese de usar semáforos globales o locales, dependiendo de lo que quiere hacer.

***Nota:** se recomienda el uso de semáforos locales cuando se necesita un semáforo para gestionar un aspecto local para un cliente de la aplicación, tales como la interfaz o un conjunto de variables interproceso. Si se utiliza un semáforo global en este caso, no sólo haría intercambios de red innecesarios, sino también podría afectar a otros equipos cliente innecesariamente. El uso de un semáforo local evitaría estos indeseables efectos secundarios.*

Sobre workers

Presentación

*Un proceso **Worker** es una forma sencilla y poderosa de intercambiar información entre procesos. Esta funcionalidad se basa en un sistema de mensajería asíncrono que permite a los procesos y formularios ser llamados y se les pide ejecutar métodos con parámetros en su propio contexto.*

***Nota:** un método de proyecto también puede ejecutarse con los parámetros en el contexto de todo formulario utilizando el comando **CALL FORM**.*

*Un worker puede ser "contratado" por cualquier proceso (utilizando el comando **CALL WORKER**) para ejecutar los métodos de proyecto con parámetros en su propio contexto, permitiendo así el acceso a información compartida.*

Esta funcionalidad responde a las siguientes necesidades en materia de comunicación entre procesos en 4D:

- *Ya que son compatibles con ambos procesos cooperativos y apropiativos, son la solución perfecta para la comunicación entre procesos en los procesos apropiativos (las variables interproceso no están permitidas en los procesos apropiativos).*
- *Ofrecen una alternativa sencilla a los semáforos, que puede ser engorrosos de definir y difíciles de usar (ver **Sobre los semáforos**).*

***Nota:** aunque se han diseñado principalmente para la comunicación entre procesos en el contexto de los procesos apropiativos (sólo en versiones 64 bits), **CALL WORKER** y **CALL FORM** están disponibles en versiones 32 bits y se pueden utilizar con los procesos cooperativos.*

Utilizar workers

*Un **worker** se utiliza para pedir a un proceso ejecutar los métodos proyecto. Un worker está hecho de:*

- *un nombre único (*), también se utiliza para nombrar su proceso asociado*
- *un proceso asociado, que puede o no puede existir en un momento dado*
- *un buzón de mensajes*
- *un método de inicio (opcional)*

(*) Advertencia: los nombres de los workers son sensibles a las mayúsculas y minúsculas ("myWorker" y "MyWorker" pueden existir simultáneamente).

*Se le pide a un worker ejecutar un método proyecto mediante una llamada al comando **CALL WORKER**. El worker y su buzón de mensajes se crean en el primer uso; su proceso asociado también se ejecuta automáticamente en el primer uso. Si el proceso worker muere luego, el buzón de mensajes permanece abierto y cualquier nuevo mensaje en el buzón iniciará un nuevo proceso worker .*

La siguiente animación ilustra esta secuencia:

*A diferencia del comando **New process**, un proceso worker permanece vivo después del final de la ejecución del método proceso. Esto significa que todas las ejecuciones del procedimiento para el mismo worker se llevarán a cabo en el mismo proceso, que mantiene toda la información de estado del proceso (variables de proceso, registro actual y selección actual ...). En consecuencia, los métodos ejecutados sucesivamente tendrán acceso y de este modo compartirán la misma información, lo que permite las comunicaciones entre procesos. El buzón de mensajes del worker maneja las llamadas sucesivas de forma asíncrona.*

***CALL WORKER** encapsula el nombre del método y los argumentos del comando en un mensaje que se publica en buzón de mensajes del worker. El proceso worker se inicia a continuación si no existe y se pide ejecutar el mensaje. Esto significa que **CALL WORKER** generalmente volverá antes de que el método se ejecute (el procesamiento es asíncrono). Por esta razón, **CALL WORKER** no devuelve ningún valor. Si necesita un worker para enviar información al proceso que lo llamó (retrollamada), es necesario utilizar **CALL WORKER** nuevamente para pasar al llamante la información necesaria. Por supuesto, en este caso, el llamante debe ser un worker.*

*No es posible utilizar **CALL WORKER** para ejecutar un método en un proceso creado por el comando **New process**. Sólo los procesos worker tienen un buzón de mensajes y, por tanto, pueden ser llamados por **CALL WORKER**. Tenga en cuenta que un proceso creado por **New process** puede llamar a un worker, pero no puede llamarse de nuevo.*

Los procesos worker se pueden crear en 4D Server a través de procedimientos almacenados: por ejemplo, puede utilizar el comando [#cmd id="373"/] para ejecutar un método que llama al comando [#cmd id="1389"/].

*Un proceso worker se cierra mediante una llamada al comando **KILL WORKER**, que vacía el buzón de mensajes del worker, le pide al proceso asociado detener el procesamiento de mensajes y terminar su ejecución actual después del final de la tarea actual.*

*El método de inicio de un worker es el método utilizado para crear el worker (como primer uso). Si **CALL WORKER** se llama con un parámetro metodo vacío, luego se reutiliza automáticamente el método de inicio como método a ejecutar.*

*El proceso principal creado por 4D cuando se abre una base para los modos aplicación y usuario es un proceso worker y puede llamarse con **CALL WORKER**. Tenga en cuenta que el nombre del proceso principal puede variar en función del lenguaje de*

*localización 4D, pero tiene siempre número de proceso 1; en consecuencia, es más conveniente para designarlo por el número de proceso en lugar del nombre del proceso al llamar **CALL WORKER**.*

Identificación de los procesos workers

*Todos los procesos workers, excepto el proceso principal, tienen el tipo Worker process (5) que devuelve el comando **PROCESS PROPERTIES**.*

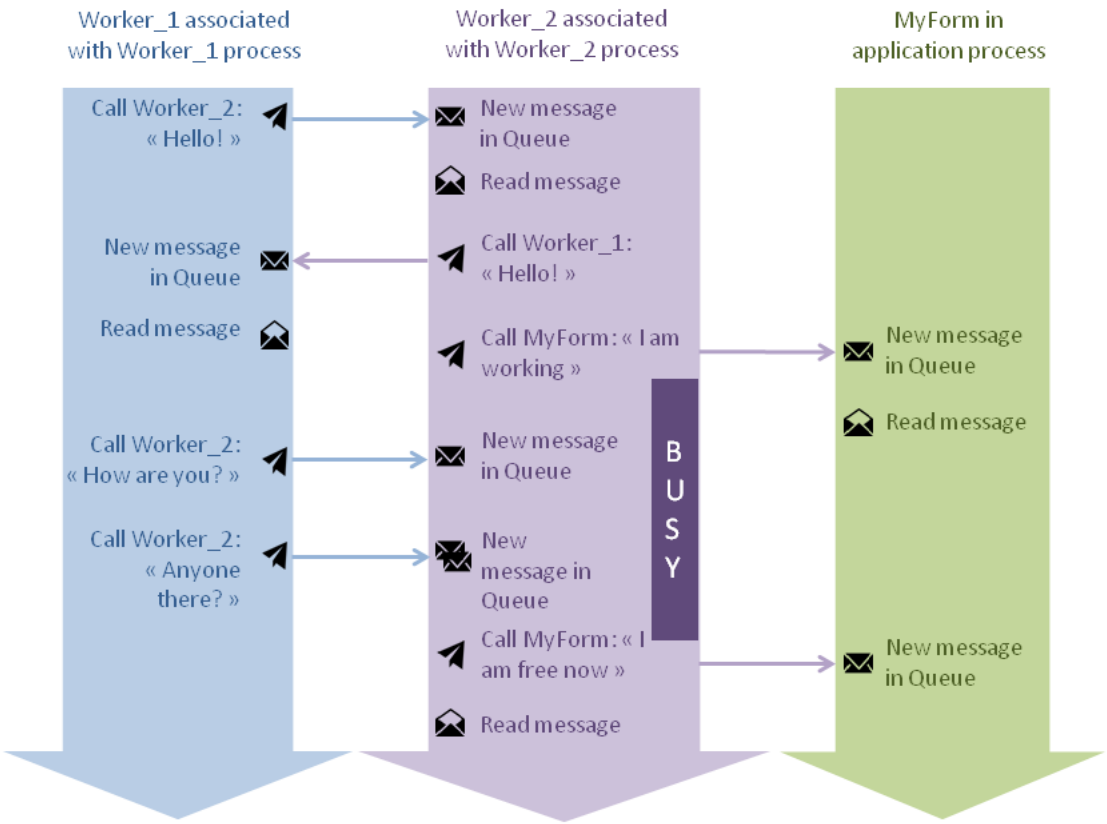
Nuevos iconos identifican los procesos workers en el Explorador de ejecución y en la ventana de administración de 4D Server:

Tipo de proceso	***Icono***
Proceso worker apropiativo	
Proceso worker cooperativo	

***Nota:** otros iconos de proceso existentes se han actualizado en 4D v15 R5.*

Principios de uso

El siguiente gráfico muestra una secuencia de comunicación entre dos workers y un formulario. La información intercambiado son sólo cadenas:



1. *Worker_1 llama a Worker_2 y pasa "Hello!" como parámetro.*
2. *Worker_2 recibe y lee el mensaje. Llama de nuevo a Worker_1 y pasa "Hello!" como parámetro.*
3. *Worker_2 a continuación, llama al formulario MyForm y pasa "I work" como parámetro. Se inicia una operación que consume tiempo y su estado se convierte en "ocupado".*
4. *Worker_1 llama a Worker_2 dos veces, pero, gracias al sistema de mensajería asíncrona, el mensaje simplemente se pone en cola. Se procesan una vez Worker_2 esté disponible, después Worker_2 ha llamado a MyForm.*

CALL WORKER

CALL WORKER (proceso ; metodo {; param}{; param2 ; ... ; paramN})

Parámetro	Tipo		Descripción
proceso	Texto, Entero largo	➡	Nombre o número del proceso worker
metodo	Texto	➡	Nombre del método proyecto a llamar
param	Expresión	➡	Parámetros pasados al método

Descripción

El comando **CALL WORKER** crea o llama al proceso worker cuyo nombre o ID se pasa en proceso y pide la ejecución del metodo en su contexto con el parámetro opcional param.

El comando **CALL WORKER** encapsula los parámetros param en un mensaje y lo envía en el buzón de mensajes del worker. Para obtener más información sobre los procesos worker, por favor consulte la sección [#title id="8727"/].

En el parámetro proceso, puede especificar el worker utilizando su nombre o su número de proceso:

- Si pasa el número de un proceso que no existe, o si el proceso especificado no fue creado por **CALL WORKER** ni por 4D (tal como el proceso principal de la aplicación), **CALL WORKER** no hace nada.
- Si pasa el nombre de un proceso que no existe, se crea un nuevo proceso worker.

Nota: el **proceso principal**, creado por 4D cuando se abre una base para la interfaz de usuario y el modo de aplicación, es un proceso workero y puede ser llamado por **CALL WORKER**.Sin embargo, ya que su nombre puede variar en función del lenguaje de 4D, es preferible designar este proceso utilizando su número (siempre 1) cuando se utiliza **CALL WORKER**.

El proceso worker aparece en la lista de procesos del Explorador de ejecución y es devuelto por el comando **PROCESS PROPERTIES** cuando se aplica a este proceso.

En metodo, se pasa el nombre del método de proyecto a ejecutar en el contexto del proceso worker. Puede pasar una cadena vacía; en este caso, el worker ejecuta el método que se utilizó originalmente para comenzar su proceso, si lo hay (es decir, el método de inicio del worker).

Nota: no es posible pasar una cadena vacía en el método cuando el comando llama al proceso principal (proceso número 1), ya que no se ha iniciado utilizando un método proyecto. Como resultado, **CALL WORKER** (1;"") no hace nada.

También puede pasar parámetros al metodo utilizando uno o más parámetros opcionales param. Pase los parámetros de la misma manera que los pasaría a una subrutina (ver la sección **Pasar parámetros a los métodos**). Al iniciar la ejecución en el contexto del proceso, el método de proceso recibe los valores de parámetro en $1, $2, y así sucesivamente. Recuerde que los arrays no pueden ser pasados como parámetros a un método. Además, en el contexto del comando **CALL WORKER**, las siguientes consideraciones adicionales deben tenerse en cuenta:

- Se permite punteros a tablas o campos.
- Los punteros a las variables, particularmente las variables locales y de proceso, no se recomiendan ya que estas variables pueden estar indefinidas en el momento de su acceso por el método de proceso.
- Si pasa un parámetro de tipo Objeto o una Colección, 4D crea una copia del objeto o de la colección en el proceso de destino si el worker se encuentra en un proceso diferente del que llama al comando **CALL WORKER**.

Un proceso worker se mantiene vivo hasta que la aplicación esté cerrada o el comando **KILL WORKER** se llame explícitamente. Para liberar memoria, no olvide llamar a este comando una vez un proceso worker ya no sea necesario.

Ejemplo

En un formulario, un botón inicia un cálculo, por ejemplo las estadísticas con respecto al año seleccionado. El botón crea o llama a un proceso de worker que va a calcular los datos mientras el usuario puede continuar trabajando en el formulario.

El método del botón es:

```
 //llamar al worker vWorkerName con el parámetro
C_LONGINT(vYear)
vYear:=2015 // podría haber sido seleccionado por el usuario en el formulario
CALL WORKER("myWorker";"workerMethod";vYear;Current form window)
```

El código de workerMethod es:

```
 // este es el método del worker
 // puede ser apropiativo o cooperativo
C_LONGINT($1) //obtiene el año
C_LONGINT($2) //obtiene la referencia de la ventana
C_OBJECT(vStatResults) //para almacenar los resultados de las estadísticas
... //calcular estadísticas
 //una vez terminado, vuelve a llamar el formulario con los valores calculados
 //vStatResults podría mostrar los resultados en el formulario
CALL FORM($2;"displayStats";vStatResults)
```

CLEAR SEMAPHORE

CLEAR SEMAPHORE (semaforo)

Parámetro	Tipo		Descripción
semaforo	Cadena	➡	Semáforo a borrar

Descripción

CLEAR SEMAPHORE borra el semaforo previamente creado por la función **Semaphore**.
La regla de utilización es que todos los semáforos deben ser borrados cuando ya no se necesitan. Si no se borran los semáforos, permanecen en memoria hasta que el proceso que los creo termine. Un proceso sólo puede borrar los semáforos que ha creado. Si usted trata de borrar un semáforo desde un proceso que no lo creo, no pasa nada.

Ejemplo

Ver el ejemplo de **Semaphore**.

GET PROCESS VARIABLE

GET PROCESS VARIABLE (proceso ; srcVar ; dstVar {; srcVar2 ; dstVar2 ; ... ; srcVarN ; dstVarN})

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número de proceso fuente
srcVar	Variable	➡	Variable fuente
dstVar	Variable	⬅	Variable de destino

Descripción

El comando ***GET PROCESS VARIABLE*** *lee el valor de las variables proceso srcVar (srvVar2, etc.) desde el proceso fuente cuyo número se pasa en proceso y devuelve sus valores actuales en las variables dstVar (dstVar2, etc.) del proceso actual.*

Cada variable fuente puede ser una variable, un array o un elemento de array. Sin embargo, tenga en cuenta las restricciones listadas más adelante en esta sección.

En cada pareja de variables srcVar;dstVar, las dos variables deben ser de tipos compatibles, de lo contrario los valores que usted obtiene podrían no ser significativos.

El proceso actual "ojea" las variables del proceso fuente, el proceso fuente no es advertido de ninguna manera de que otro proceso está leyendo la instancia de sus variables.

4D Server: *utilizando 4D Client, puede leer las variables en un proceso de destino ejecutado en el equipo servidor (procedimiento almacenado). Para hacer esto, coloque un signo menos antes del número de identificación del proceso en el parámetro proceso.*

La comunicación proceso "Intermachine", ofrecida por los comandos ***GET PROCESS VARIABLE****,* ***SET PROCESS VARIABLE*** *y* ***VARIABLE TO VARIABLE****, es posible únicamente desde el cliente al servidor. Siempre es un proceso cliente el que lee o escribe las variables de un procedimiento almacenado.*

Tip: *si no conoce el número de identificación del proceso servidor, aún puede utilizar las variables interproceso del servidor. Para hacer esto, puede utilizar cualquier valor negativo en proceso. En otras palabras, no es necesario conocer el número de identificación del proceso para poder utilizar el comando* ***GET PROCESS VARIABLE*** *con las variables interproceso del servidor. Esta posibilidad es muy útil particularmente cuando un procedimiento almacenado se lanza utilizando el método base* ***On server startup****. Como los equipos cliente no conocen automáticamente el número de identificación de ese proceso, todo valor negativo puede pasarse en el parámetro proceso.*

Restricciones

GET PROCESS VARIABLE *no acepta variables locales como variables fuente.*

Por otra parte, las variables de destino pueden ser interproceso, proceso o locales. Los valores se "reciben" únicamente en las variables, no en los campos.

GET PROCESS VARIABLE *acepta todo tipo de variable fuente, proceso o interproceso, excepto:*

- *Punteros*
- *Array de punteros*
- *Arrays de dos dimensiones*

El proceso fuente debe ser un proceso usuario; no puede ser un proceso kernel. Si el proceso fuente no existe, este comando no tiene efecto.

Nota: *en modo interpretado, si una variable fuente no existe, se devuelve el valor indefinido. Puede detectar esto utilizando la función* ***Type*** *para probar la variable de destino correspondiente.*

Ejemplo 1

Esta línea de código lee el valor de la variable texto vtEstadoActual en el proceso cuyo número es $vlProceso y devuelve el resultado en la variable proceso vtInfo del proceso actual:

```
GET PROCESS VARIABLE($vlProceso;vtEstadoActual;vtInfo)
```

Ejemplo 2

Esta línea de código hace lo mismo, pero devuelve el valor en la variable local $vtInfo de método que se está ejecutando en el proceso actual:

```
GET PROCESS VARIABLE($vlProceso;vtEstadoActual;$vtInfo)
```

Ejemplo 3

Esta línea de código hace lo mismo pero devuelve el valor en la variable vtEstadoActual del proceso actual:

```
GET PROCESS VARIABLE($vlProceso;vtEstadoActual;vtEstadoActual)
```

Nota: *el primer vtCurStatus designa la instancia de la variable en el proceso fuente, el segundo vtCurStatus designa la instancia de la variable en el proceso actual.*

Ejemplo 4

Este ejemplo lee secuencialmente los elementos de un array proceso desde el proceso indicado por $vlProceso:

```
GET PROCESS VARIABLE($vlProceso;vl_IPCom_Array;$vlTam)
For($vlElem;1;$vlTam)
   GET PROCESS VARIABLE($vlProceso;at_IPCom_Array{$vlElem};$vtElem)
  ` Hacer algo con $vtElem
End for
```

***Nota:** en este ejemplo, la variable proceso vl_IPCom_Array contiene el tamaño del Array at_IPCom_Array, y debe ser mantenida por el proceso fuente.*

Ejemplo 5

Este ejemplo hace lo mismo que el anterior, pero lee el array como un todo, en lugar de leer los elementos de manera secuencial:

```
GET PROCESS VARIABLE($vlProceso;at_IPCom_Array;$anArray)
For($vlElem;1;Size of array($anArray))
  ` Hacer algo con $anArray{$vlElem}
End for
```

Ejemplo 6

Este ejemplo lee las instancias de las variables v1,v2,v3 en el proceso fuente y devuelve sus valores en la instancia de las mismas variables del proceso actual:

```
GET PROCESS VARIABLE($vlProceso;v1;v1;v2;v2;v3;v3)
```

Ejemplo 7

*Ver el ejemplo del comando **DRAG AND DROP PROPERTIES**.*

KILL WORKER

KILL WORKER {(proceso)}

Parámetro	Tipo		Descripción
proceso	Texto, Entero largo	➡	Número o nombre del proceso a matar (proceso actual si se omite)

Descripción

*El comando **KILL WORKER** envía un mensaje al proceso worker cuyo nombre o número que pasa en proceso, pidiéndole hacer caso omiso de los mensajes pendientes y poner fin a su ejecución tan pronto como terminen las tareas actuales.*

*Este comando sólo se puede utilizar con los procesos worker. Para más información, por favor consulte la sección **Sobre workers**.*

*En proceso, se pasa el nombre o el número del proceso worker cuya ejecución debe terminarse. Si no existe ningún worker con el nombre o el número de proceso especificado, **KILL WORKER** no hace nada.*
*Si no pasa ningún parámetro, **KILL WORKER** aplica a los procesos worker actualmente en ejecución y por lo tanto es equivalente a **KILL WORKER** (Current process).*

*Si **KILL WORKER** se aplica a un worker que no fue creado explícitamente utilizando el comando **CALL WORKER** (por ejemplo, el proceso worker principal de la aplicación), sólo pide a este worker vaciar su buzón de mensajes.*

*Si se llama al comando **CALL WORKER** para enviar un mensaje a un worker que acaba de ser finalizado por **KILL WORKER**, se inicia un nuevo proceso. Para asegurarse de que solo hay un proceso que se ejecuta a la vez para un trabajador, el nuevo proceso se iniciará una vez que el anterior haya finalizado. Sin embargo, tenga en cuenta que si **CALL WORKER** es llamado por un worker para enviarse un mensaje, mientras que acaba de ser eliminado por **KILL WORKER**, el comando no hace nada.*

Ejemplo

El siguiente código (ejecutado desde un formulario, por ejemplo) dará lugar a la terminación del worker:

```
CALL WORKER(vWorkerName;"theWorker";"end")
```

En el método (theWorker), se añade código para manejar esta situación:

```
 //método theWorker
C_TEXT($1) //param

Case of
   :($1="call") //se llama al worker
      ... //do something
   :($1="end") //se le pide al worker suicidarse
      KILL WORKER
End case
```

Semaphore

Semaphore (semaforo {; ticCont}) -> Resultado

Parámetro	Tipo		Descripción
semaforo	Cadena	➡	Semáforo a probar y posicionar
ticCont	Entero largo	➡	Máximo tiempo de espera
Resultado	Booleano	⮌	El semáforo se ha creado correctamente (FALSE) o El semáforo ya había sido creado (TRUE)

Descripción

Un semáforo es una bandera compartida entre estaciones de trabajo o entre procesos en la misma estación de trabajo. Un semáforo simplemente existe o no existe. Los métodos que cada usuario ejecuta pueden probar la existencia de un semáforo. Un semáforo sólo puede ser eliminado por la estación de trabajo cliente o proceso que lo creó. Creando y probando semáforos, los métodos pueden comunicarse entre estaciones de trabajo. No utilice semáforos para proteger el acceso a los registros. Esto lo hace automáticamente 4D y 4D Server. Utilice los semáforos para evitar que varios usuarios realice la misma operación al mismo tiempo.

La función ***Semaphore*** *devuelve TRUE y no hace nada si el semaforo existe. Si semaforo no existe,* ***Semaphore*** *lo crea y devuelve FALSE. Sólo un usuario a la vez puede crear un semáforo. Si semaforo devuelve FALSE, esto significa que el semáforo no existe, pero también significa que el semáforo ha sido creado para el proceso en el cual la llamada se ha efectuado.*

Semaphore *devuelve FALSE si el semáforo no fue definido. También devuelve FALSE si el semáforo ya fue definido por el mismo proceso en el cual la llamada ha sido efectuada. Un semáforo está limitado a 255 caracteres, incluyendo el prefijo ($). Si pasa una cadena más larga, el semáforo se probará con la cadena truncada.*

Recuerde que los nombres de los semáforos son sensibles a las mayúsculas y minúsculas en 4D (por ejemplo, el programa considera que "MySemaphore" es diferente de "mysemaphore"

El parámetro opcional ticCont le permite especificar un tiempo de espera (en tics) si semaforo ya está definido. En este caso, la función esperará o que el semáforo sea liberado o el tiempo de espera para terminar antes de devolver ***True****.*

Hay dos tipos de semáforos en 4D: semáforos locales y semáforos globales.

• Un semáforo local es visible para todos los procesos de una misma estación de trabajo y solamente en la estación de trabajo. Declare un semáforo local colocando al nombre del semáforo el signo dólar ($) como prefijo. Los semáforos locales permiten controlar las operaciones entre los diferentes procesos ejecutados en la misma estación de trabajo. Por ejemplo, un semáforo local puede ser utilizado para monitorear los accesos a un array interproceso compartido por todos los procesos de una base mono usuario o de un equipo cliente.
• Un semáforo global es visible para todos los usuarios y todos los procesos. Los semáforos globales permiten controlar las operaciones entre los equipos clientes de una base multiusuarios.

Los semáforos globales y locales son idénticos en su lógica. La diferencia reside en su alcance, es decir en su visibilidad. En cliente-servidor, los semáforos globales son visibles para todos los procesos de todos los equipos clientes y del servidor. Un semáforo local sólo es visible para los procesos del equipo en el que ha sido creado.
Con 4D, los semáforos globales y locales tienen el mismo alcance porque usted es el único usuario. Sin embargo, si su base está siendo utilizada en los dos entornos, asegúrese de utilizar semáforos globales y locales dependiendo de lo que quiera hacer.

Nota: *los semáforos locales se recomiendan cuando el uso de un semáforo es necesario para manejar un aspecto local para un cliente de la aplicación, tal como la interfaz o un array de valores interproceso. El uso de un semáforo global provocará en este caso, no sólo intercambios de red innecesarios, sino también puede afectar otras máquinas cliente. El uso de un semáforo local evitará estos efectos indeseables.*

Ejemplo 1

Este es el código típico para utilizar un semáforo:

```
While(Semaphore("MySemaphore";300))
   IDLE
End while
 // coloque el código protegido por el semáforo aquí
CLEAR SEMAPHORE("MySemaphore")
```

Ejemplo 2

En este ejemplo, usted quiere evitar que dos usuarios efectúen simultáneamente una actualización global de los precios en una tabla Productos. El siguiente método utiliza semáforos para hacer esto:

```
If(Semaphore("ActualizacionPrecios"))  ` Trate de crear el semáforo
   ALERT("Otro usuario ya está actualizando los precios. Inténtelo más tarde.")
Else
   ActualizarPrecios  ` Actualización de todos los precios
   CLEAR SEMAPHORE("ActualizacionPrecios")  ` Borrar el semáforo
End if
```

Ejemplo 3

El siguiente ejemplo utiliza un semáforo local. En una base con varios procesos, usted quiere mantener una lista de "Cosas por hacer". Usted quiere mantener la lista en un array interproceso y no en una tabla. Usted utiliza un semáforo para evitar el acceso simultáneo. En esta situación, sólo necesita utilizar un semáforo local, porque su lista "Cosas por hacer" es sólo para su uso personal.

El array interproceso se inicializa en el método Startup:

```
ARRAY TEXT(◇ListaHacer;0)  ` La lista de cosas por hacer está inicialmente vacía
```

Este es el método utilizado para añadir elementos a la lista de cosas por hacer:

```
  ` Método de proyecto AGREGAR A LISTA DE COSAS POR HACER
  ` AGREGAR A LISTA DE COSAS POR HACER ( Texto )
  ` AGREGAR A LISTA DE COSAS POR HACER ( Elemento de la lista de cosas por hacer)
C_TEXT($1)
If(Not(Semaphore("$AccesoLista";300)))
  ` Espera 5 segundos si el semáforo ya existe
   $vlElem:=Size of array(◇ListaHacer)+1
   INSERT IN ARRAY(◇ListaHacer;$vlElem)
   ◇ListaHacer{$vlElem}:=$1
   CLEAR SEMAPHORE("$AccesoLista")  ` Borrar el semáforo
End if
```

Puede llamar este método desde cualquier proceso.

Ejemplo 4

Este método permite no ejecutar un método si el semáforo está presente; el método informa el método de llamada con un código de error y un texto plano.

Sintaxis:

```
$L_Error:=Semaphore_proof(->$T_Text_error)
```

```
  // Estructura de protección por semáforo
C_LONGINT($0)
C_POINTER($1)  // mensaje de error

  // Inicio del método
C_LONGINT($L_MyError)
$L_MyError:=1

C_TEXT($T_Sema_local)
$T_Sema_local:="$tictac"

If(Semaphore($T_Sema_local;300))
  // Esperábamos 300 tics pero el semáforo
  // no fue lanzado por el que lo ubicó:
  // terminamos aquí
   $L_MyError:=-1

Else

  // Este método solo se ejecuta por un proceso a la vez

  // Ubicamos el semáforo al mismo tiempo que lo introducimos
  // así que somos los únicos que lo podemos eliminar

  // Hacer algo
   ...
  // Terminar eliminando el semáforo
   CLEAR SEMAPHORE($T_Sema_local)
End if

C_TEXT($T_Message)
If($L_MyError=-1)
   $T_Message:="El semáforo "+$T_Sema_local+" tiene el acceso bloqueado al resto del código"
Else
   $T_Message:="OK"
End if

$0:=$L_MyError
$1->:=$T_Message  // El método llamante recibe un código de error y una explicación en texto plano
```

SET PROCESS VARIABLE

SET PROCESS VARIABLE (proceso ; dstVar ; expr {; dstVar2 ; expr2 ; ... ; dstVarN ; exprN})

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número de proceso de destino
dstVar	Variable	➡	Variable de destino
expr	Variable	➡	Expresión fuente (o variable fuente)

Descripción

El comando ***SET PROCESS VARIABLE*** *escribe las variables proceso dstVar (dstVar2, etc.) del proceso de destino cuyo número se pasa en proceso utilizando los valores pasados en expr1 (expr2, etc.).*

Cada variable de destino puede ser una variable o un elemento de array. Sin embargo, vea las restricciones listadas más adelante en esta sección.

Para cada pareja de variables dstVar;expr, la expresión debe ser compatible con la variable de destino, de lo contrario usted puede terminar con un valor en la variable que no tiene significado. En modo interpretado, si una variable de destino no existe, se crea y asigna con la expresión.

El proceso actual escribe las variables del proceso de destino, el proceso de destino no es advertido de ninguna manera de que otro proceso está escribiendo la instancia de sus variables.

4D Server: *utilizando 4D Client, puede escribir variables en un proceso de destino ejecutado en el equipo servidor (procedimiento almacenado). Para hacer esto, coloque un signo menos antes del número de identificación del proceso en el parámetro proceso.*
La comunicación proceso "Intermachine", ofrecida por los comandos ***GET PROCESS VARIABLE****,* ***SET PROCESS VARIABLE*** *y* ***VARIABLE TO VARIABLE****, es posible únicamente desde el cliente al servidor. Siempre es un proceso cliente el que lee o escribe las variables de un procedimiento almacenado.*

Tip: *en otras palabras, no es necesario conocer el número de identificación del proceso para poder utilizar el comando* ***SET PROCESS VARIABLE*** *con las variables interproceso del servidor. Esta posibilidad es muy útil particularmente cuando un procedimiento almacenado se lanza utilizando el método base* ***On server startup****. Como los equipos cliente no conocen automáticamente el número de identificación de ese proceso, todo valor negativo puede pasarse en el parámetro proceso.*

Restricciones

SET PROCESS VARIABLE *no acepta variables locales como variables de destino.*

SET PROCESS VARIABLE *acepta todo tipo de proceso de variable proceso o interproceso de destino, excepto:*

- *Punteros*
- *Arrays de todo tipo. Para escribir un array como un todo de un proceso a otro, utilice el comando* ***VARIABLE TO VARIABLE****. Nota, sin embargo,* ***SET PROCESS VARIABLE*** *le permite escribir el elemento de un array.*
- *No es posible escribir el elemento de un array de punteros o el elemento de un array de dos dimensiones.*

El proceso de destino debe ser un proceso usuario; no puede ser un proceso kernel. Si el proceso de destino no existe, se genera un error. Puede encontrar este error utilizando un método de gestión de errores instalado con ***ON ERR CALL****.*

Ejemplo 1

La siguiente línea de código asigna (a la cadena vacía) el texto de la variable vtEstadoActual del proceso cuyo número es $vlProceso:

```
SET PROCESS VARIABLE($vlProceso;vtEstadoActual;"")
```

Ejemplo 2

Esta línea de código asigna a la variable texto vtEstadoActual del proceso cuyo número es $vlProceso el valor de la variable $vtInfo desde el método de ejecución en el proceso actual:

```
SET PROCESS VARIABLE($vlProceso;vtEstadoActual;$vtInfo)
```

Ejemplo 3

Esta línea de código define el texto de la variable vtEstadoActual del proceso cuyo número es $vlProceso al valor de la misma variable en el proceso actual:

```
SET PROCESS VARIABLE($vlProceso;vtEstadoActual;vtEstadoActual)
```

Nota: *el primer vtEstadoActual designa la instancia de la variable en el proceso de destino. El segundo vtEstadoActual designa la instancia de la variable en el proceso actual.*

Ejemplo 4

Este ejemplo vuelve mayúsculas secuencialmente todos los elementos de un array proceso desde el proceso indicado por $vlProceso:

```
GET PROCESS VARIABLE($vlProceso;vl_IPCom_Array;$vlTam)
For($vlElem;1;$vlSize)
    GET PROCESS VARIABLE($vlProceso;at_IPCom_Array{$vlElem};$vtElem)
    SET PROCESS VARIABLE($vlProceso;at_IPCom_Array{$vlElem};Uppercase($vtElem))
End for
```

Nota: *en este ejemplo, la variable proceso vl_IPCom_Array contiene el tamaño del array at_IPCom_Array y debe ser mantenida por el proceso fuente/destino.*

Ejemplo 5

Este ejemplo escribe la instancia de las variables v1, v2 y v3 utilizando la instancia de las mismas variables desde el proceso actual:

```
SET PROCESS VARIABLE($vlProceso;v1;v1;v2;v2;v3;v3)
```

Test semaphore

Test semaphore (semaforo) -> Resultado

Parámetro	Tipo		Descripción
semaforo	Cadena	→	Nombre del semáforo a probar
Resultado	Booleano	←	True = el semáforo existe, False = el semáforo no existe

Descripción

El comando **Test semaphore** le permite probar la existencia de un semáforo.

La diferencia entre la función **Semaphore** y **Test semaphore** es que **Test semaphore** no crea el semaphore si éste no existe. Si el semaforo existe, la función devuelve **True**. De lo contrario, devuelve **False**.

Ejemplo

El siguiente ejemplo le permite conocer el estado de un proceso (en nuestro caso, la modificación de un código) sin modificar semaforo:

```
$Win:=Open window(x1;x2;y1;y2;-Palette window)
Repeat
   If(Test semaphore("Código de encriptación"))
      POSICION MENSAJE($x3;$y3)
      MESSAGE("El código de encriptación está siendo modificado.")
   Else
      POSICION MENSAJE($x3;$y3)
      MESSAGE("La modificación del código de encriptación ha sido autorizada.")
   End if
Until(StopInfo)
CLOSE WINDOW
```

VARIABLE TO VARIABLE

VARIABLE TO VARIABLE (proceso ; dstVar ; srcVar {; dstVar2 ; srcVar2 ; ... ; dstVarN ; srcVarN})

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número de proceso de destino
dstVar	Variable	➡	Variable de destino
srcVar	Variable	➡	Variable fuente

Descripción

El comando ***VARIABLE TO VARIABLE*** *escribe las variables proceso dstVar (dstVar2, etc.) del proceso de destino cuyo número se pasa en proceso utilizando los valores de las variables srcVar1 srcVar2, etc.*

VARIABLE TO VARIABLE *tiene la misma acción de* ***SET PROCESS VARIABLE****, con las siguientes diferencias:*
• Usted pasa expresiones fuente a ***SET PROCESS VARIABLE*** *y por lo tanto no puede pasar un array como un todo. Debe pasar exclusivamente variables fuente a* ***VARIABLE TO VARIABLE*** *y por lo tanto puede pasar un array como un todo.*
• Cada variable de destino puede ser una variable o un elemento de un array, pero no puede ser un array como un todo. Cada variable de destino de ***VARIABLE TO VARIABLE*** *puede ser una variable, un array o un elemento de array.*

4D Server: *la comunicación proceso "Intermachine", ofrecida por los comandos* ***GET PROCESS VARIABLE****,* ***SET PROCESS VARIABLE*** *y* ***VARIABLE TO VARIABLE****, es posible únicamente desde el cliente al servidor. Siempre es un proceso cliente el que lee o escribe las variables de un procedimiento almacenado.*

Para cada pareja de variables dstVar;expr, la variable fuente debe ser de tipo compatible con la variable de destino, de lo contrario puede terminar con un valor en la variable que no tiene significado. En modo interpretado, si una variable de destino no existe, se crea y asigna con el tipo y valor de la variable fuente.

El proceso actual escribe las variables del proceso de destino, el proceso de destino no es advertido de ninguna manera de que otro proceso está escribiendo la instancia de sus variables.

Restricciones

VARIABLE TO VARIABLE *no acepta variables locales como variables de destino.*

VARIABLE TO VARIABLE *acepta todo tipo de variable proceso o interproceso de destino, a excepción de las variables de tipo:*

- *Punteros*
- *Arrays de punteros*
- *Arrays de dos dimensiones*

El proceso de destino debe ser un proceso usuario; no puede ser un proceso kernel. Si el proceso de destino no existe, se genera un error. Usted puede encontrar este error utilizando un método de gestión de errores instalado con ***ON ERR CALL****.*

Ejemplo

El siguiente ejemplo lee un array proceso desde el proceso indicado por $vlProceso, secuencialmente convierte los elementos a mayúsculas y luego escribe completamente el array:

```
GET PROCESS VARIABLE($vlProceso;at_IPCom_Array;$anArray)
For($vlElem;1;Size of array($anArray))
   $anArray{$vlElem}:=Uppercase($anArray{$vlElem})
End for
VARIABLE TO VARIABLE($vlProceso;at_IPCom_Array;$anArray)
```

Procesos (Interfaz de usuario)

- *BRING TO FRONT*
- *Frontmost process*
- *HIDE PROCESS*
- *SHOW PROCESS*

BRING TO FRONT

BRING TO FRONT (proceso)

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número del proceso a pasar al primer plano

Descripción

BRING TO FRONT *pasa todas las ventanas que pertenecen a process al primer plano. Si el proceso ya está en el primer plano, el comando no hace nada. Si el proceso está oculto, debe utilizar* ***SHOW PROCESS*** *para mostrar el proceso, de lo contrario* ***BRING TO FRONT*** *no tiene efecto.*

Los procesos Principal y Diseño pueden pasarse al primer plano utilizando este comando.

Nota: *cuando el proceso contiene varias ventanas y quiere pasar al primer plano una ventana específica, es preferible utilizar por ejemplo, el comando* ***SET WINDOW RECT****.*

Ejemplo

El siguiente ejemplo es un método que puede ser ejecutado desde un menú. Él verifica si el proceso del primer plano es el proceso <>Clientes. Si no, el método lo pasa al primer plano:

```
If(Frontmost process(◇Clientes)
    BRING TO FRONT(◇Clientes)
End if
```

Frontmost process

Frontmost process {(*)} -> resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Número del proceso de la primera ventana no flotante
resultado	Entero	⮌	Número del proceso cuyas ventanas están en el primer plano

Descripción

Frontmost process *devuelve el número del proceso cuya ventana (o ventanas) están en el primer plano.*
Cuando tiene una o más ventanas flotantes abiertas, hay dos capas de ventanas:

- *Ventanas estándar*
- *Ventanas flotantes*

Si la función ***Frontmost process*** *se utiliza en el método de formulario o en un método de objeto de una ventana flotante, la función devuelve el número de referencia de la ventana flotante que se encuentra más adelante en la capa de ventanas flotantes. Si especifica el parámetro opcional *, la función devuelve el número de referencia del proceso de la ventana activa que se encuentra en el primer plano en las capas de ventanas estándar.*

Ejemplo

Ver el ejemplo para ***BRING TO FRONT****.*

HIDE PROCESS

HIDE PROCESS (proceso)

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número de proceso a ocultar

Descripción

HIDE PROCESS oculta todas las ventanas que pertenecen a proceso. Todos los elementos de interfaz de proceso se ocultan hasta el siguiente **SHOW PROCESS**. La barra de menús del proceso también se oculta. Esto significa que la apertura de una ventana mientras el proceso está oculto no provocará ningún cambio en la visualización en pantalla. Si el proceso ya está oculto, el comando no tiene ningún efecto.

La única excepción a esta regla es la ventana del depurador. Si la ventana del depurador se muestra cuando proceso es un proceso oculto, proceso se muestra y pasa al primer plano del proceso.

Si no quiere que un proceso se muestre cuando es creado, **HIDE PROCESS** debe ser el primer comando en el método de proceso. Los procesos Usuario/Menús personalizados y los procesos de gestión de la caché no pueden ocultarse utilizando este comando.

Aunque un proceso esté oculto, el proceso está aún en ejecución.

Ejemplo

El siguiente ejemplo oculta todas las ventanas que pertenecen al proceso actual:

```
HIDE PROCESS(Current process)
```

SHOW PROCESS

SHOW PROCESS (proceso)

Parámetro	Tipo		Descripción
proceso	Entero largo	➡	Número de proceso del proceso a mostrar

Descripción

SHOW PROCESS *muestra todas las ventanas que pertenecen a proceso. Este comando no trae las ventanas de proceso al primer plano. Para hacer esto, utilice el comando* ***BRING TO FRONT****.*
Si el proceso ya está siendo visualizado, el comando no tiene efecto.

Ejemplo

El siguiente ejemplo muestra un proceso llamado Clientes, que se ha ocultado previamente. El número de proceso se almacena en la variable interproceso <>Clientes:

```
SHOW PROCESS(◇Clientes)
```

Protocolo seguro

GENERATE CERTIFICATE REQUEST
GENERATE ENCRYPTION KEYPAIR

GENERATE CERTIFICATE REQUEST

GENERATE CERTIFICATE REQUEST (llavePriv ; peticionCertif ; arrayCod ; arrNombres)

Parámetro	Tipo		Descripción
llavePriv	BLOB	➡	BLOB que contiene la llave privada
peticionCertif	BLOB	⬅	BLOB que recibe la solicitud del certificado
arrayCod	Array entero largo	➡	La lista de códigos de información
arrNombres	Array cadena	➡	Lista de nombres

Descripción

*El comando **GENERATE CERTIFICATE REQUEST** genera una solicitud de certificación al formato PKCS el cual puede ser utilizado directamente por las autoridades de certificación tal como Verisign(R). El certificado es una parte importante en el protocolo seguro SSL. El certificado se envía a cada navegador que se conecta en modo SSL y contiene la "tarjeta de identidad" del sitio web (con la información introducida en el comando), así como también su llave pública permitiendo a los navegadores descifrar la información recibida. Además, el certificado contiene diferente información añadida por la autoridad de certificación lo cual garantiza su integridad.*

***Nota:** para mayor información sobre el protocolo SSL utilizado con el servidor web 4D, consulte la sección **Utilizar el protocolo TLS (HTTPS)**.*

*La solicitud de certificación utiliza un par de llaves generadas por el comando **GENERATE ENCRYPTION KEYPAIR** y contiene diferente información. La autoridad de certificación generará su certificado combinando esta solicitud con otros parámetros.*

*Pase en llavePriv un BLOB que contenga la llave privada genera con el comando **GENERATE ENCRYPTION KEYPAIR**.*

*Pase en peticionCertif un BLOB vacío. Una vez el comando ha sido ejecutado, contiene la solicitud de certificación al formato PKCS codificada en base64. Puede almacenar esta solicitud en un archivo de texto, por ejemplo utilizando el comando **BLOB TO DOCUMENT**, para presentarlo a la autoridad de certificación.*

***Advertencia:** la llave privada se utiliza para generar la solicitud de certificación pero NO debe ser enviada a la autoridad de certificación.*

Los arrays arrayCod (entero largo) y arrayNom (cadena) deben llenarse con los números de código y la información requerida por la autoridad de certificación respectivamente.

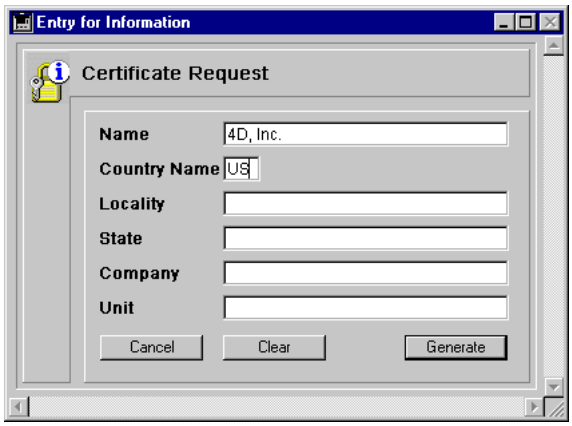
Los códigos y nombres requeridos pueden cambiar de acuerdo a la autoridad de certificación y el uso del certificado. Sin embargo, dentro del uso normal del certificado (conexiones del servidor web vía SSL), los arrays deben contener los siguientes elementos:

Información a suministrar	arrayCod	arrayNom (Ejemplos)
Nombre del dominio	13	www.4dhispano.com
Código del país (dos letras)	14	ES
Ciudad	15	Barcelona
Estado, Departamento,...	16	Cataluña
Nombre de la organización	17	4D Hispano
Servicio/Persona a cargo del servidor	18	Administrador Web

*El orden en el que se introducen los códigos y la información no importa, sin embargo los dos arrays deben estar sincronizados: si el tercer elemento de codeArray contiene el valor 15 (ciudad), el tercer elemento de nameArray debe contener esta información, en nuestro ejemplo **Barcelona**.*

Ejemplo

*Un formulario "Solicitud de certificado" contiene los seis campos necesarios para una solicitud de certificación estándar. El botón **Generar** crea un documento en disco que contiene la solicitud del certificado. El documento "Privatekey.txt" que contiene la llave privada (generada con el comando **GENERATE ENCRYPTION KEYPAIR**) debe estar en el disco:*



*Este es el método del botón **Generar***

```
` Método de objeto bGenerar
```

```
C_BLOB($vbllavePriv;$vbpeticionCertif)
C_LONGINT($tablaNum)
ARRAY LONGINT($tLCodigos;6)
ARRAY STRING(80;$tSInfos;6)

$tableNum:=Table(Current form table)
For($i;1;6)
    $tSInfos{$i}:=Field($tablaNum;$i)->
    $tLCodigoss{$i}:=$i+12
End for
If(Find in array($tSInfos;"")#-1)
    ALERT("Todos los campos deben ser completados.")
Else
    ALERT("Seleccione su llave privada.")
    $vhDocRef:=Open document("")
    If(OK=1)
        CLOSE DOCUMENT($vhDocRef)
        DOCUMENT TO BLOB(Document;$vbllavePriv)
        GENERATE CERTIFICATE REQUEST($vbllavePriv;$vbcertifRequest;$tLCodigos;$tSInfos)
        BLOB TO DOCUMENT("Request.txt";$vbcertifRequest)
    Else
        ALERT("Llave privada inválida.")
    End if
End if
```

GENERATE ENCRYPTION KEYPAIR

GENERATE ENCRYPTION KEYPAIR (llavePriv ; llavepublica {; longitud})

Parámetro	Tipo		Descripción
llavePriv	BLOB	⇐	BLOB que contiene la llave privada
llavepublica	BLOB	⇐	BLOB que contiene la llave pública
longitud	Entero largo	⇒	Longitud de la llave (bits) [386...2048] Valor por defecto = 512

Descripción

*El comando **GENERATE ENCRYPTION KEYPAIR** genera un nuevo par de llaves RSA. El sistema de seguridad ofrecido en 4D está basado en llaves destinadas a cifrar/descifrar información. Las llaves pueden ser utilizadas dentro del protocolo TSL/SSL, con el servidor web 4D (cifrado y seguridad de las comunicaciones) y en todas las bases de datos (cifrado de datos).*

Una vez ejecutado el comando, los BLOBs pasados en los parámetros llavePriv y llavepublica contienen un nuevo par de llaves de cifrado.

El parámetro opcional largo permite precisar el tamaño de la llave (en bits). Entre más larga sea la llave, más difícil es romper el código cifrado.

Sin embargo, las llaves largas necesitan más tiempo de ejecución o de respuesta, especialmente dentro de una conexión segura.

Por defecto (si omite el parámetro largo), el tamaño de llave generado es de 512 bits, lo cual es un buen compromiso para la relación seguridad/eficiencia. Para aumentar el factor de seguridad, puede cambiar las llaves con frecuencia, por ejemplo cada seis meses. Puede generar llaves de 2048 bits para aumentar la seguridad de cifrado pero podría disminuir la velocidad de las conexiones a la aplicación web.
*Este comando genera llaves en formato PKCS codificadas en base64, lo que significa que su contenido puede copiarse y pegarse en un correo electrónico sin sufrir ningún cambio. Una vez se genera el par de llaves, se puede generar un documento de texto en formato PEM (utilizando por ejemplo el comando **BLOB TO DOCUMENT**) y las llaves pueden guardarse en un lugar seguro.*

***Advertencia:** la llave privada siempre debe mantenerse en secreto.*

RSA, llaves privadas y llaves públicas

*El algoritmo de cifrado RSA utilizado por **GENERATE ENCRYPTION KEYPAIR** está basado en un sistema de cifrado de doble llave: una llave privada y una llave pública. Como su nombre lo indica, la llave pública puede ser entregada a una tercera persona y utilizada para descifrar la información. La llave pública corresponde a una llave privada única, utilizada para cifrar la información. De esta forma, la llave privada se utiliza para el cifrado; la llave pública para descifrar (o viceversa). La información cifrada con una llave sólo puede ser descifrada con la otra.*

*Las funciones de cifrado del protocolo TLS/SSL están basadas en este principio, la llave pública que se incluye en el certificado enviado a los navegadores (para mayor información, consulte la sección **Utilizar el protocolo TLS (HTTPS)**).*

*Este modo de cifrado también lo utiliza la primera sintaxis de los comandos **ENCRYPT BLOB** y **DECRYPT BLOB**. La llave pública debe ser publicada de manera confidencial.*

*Es posible combinar las llaves públicas y privadas de dos personas para cifrar información de manera que el receptor sea la única persona que pueda descifrar los datos y el emisor la única persona que puede cifrarlos. Es el principio de la segunda sintaxis de los comandos **ENCRYPT BLOB** y **DECRYPT BLOB**.*

Ejemplo

*Ver el ejemplo del comando **ENCRYPT BLOB**.*

Recursos

- Recursos
- CLOSE RESOURCE FILE
- GET ICON RESOURCE
- Get indexed string
- GET PICTURE RESOURCE
- GET RESOURCE
- Get resource name
- Get resource properties
- Get string resource
- Get text resource
- Open resource file
- RESOURCE LIST
- RESOURCE TYPE LIST
- STRING LIST TO ARRAY
- _o_ARRAY TO STRING LIST
- _o_Create resource file
- _o_DELETE RESOURCE
- _o_Get component resource ID
- _o_SET PICTURE RESOURCE
- _o_SET RESOURCE
- _o_SET RESOURCE NAME
- _o_SET RESOURCE PROPERTIES
- _o_SET STRING RESOURCE
- _o_SET TEXT RESOURCE

Recursos

Notas de compatibilidad sobre la escritura de recursos (4D v13)

Como se anunció con el lanzamiento de 4D v11 (ver la sección a continuación), los mecanismos basados en el uso de archivos de recursos son obsoletos. ***Los comandos del tema "Resources" utilizados para escribir en los archivos de recursos no se deben utilizar más; serán removidos de las versiones futuras de 4D.*** *Los comandos utilizados para leer recursos se mantienen por compatibilidad.*

Notas de compatibilidad sobre los mecanismos de gestión de recursos (4D v11)

La gestión de recursos se ha modificado en 4D a partir de la v11. Conforme a las direcciones especificadas por Apple e implementadas en las versiones Mac OS más recientes, el concepto de recursos en el sentido más estricto (ver la definición más adelante) ahora es obsoleto y se abandonará progresivamente. Se han implementado nuevos mecanismos para soportar las necesidades que antes eran cubiertas por recursos: archivos XLIFF para la traducción de cadenas de caracteres, archivos de imágenes .png... De hecho, los archivos de recursos se reemplazarán a favor de los archivos de tipo estándar. 4D acompaña esta evolución y a partir de la versión 11, ofrece nuevas herramientas para la gestión de traducciones de bases de datos, mientras mantiene la compatibilidad con los sistemas existentes.

Compatibilidad

Para mantener la compatibilidad y con el fin de permitir la adaptación progresiva a las aplicaciones existentes, los mecanismos de gestión de recursos continúan funcionando en 4D v11, con unas pequeñas diferencias:

- *Cuando están presentes, los archivos de recursos aún son soportados por 4D y el principio de la "cadena de archivos de recursos" (apertura sucesiva de varios archivos de recursos) continúa siendo válida. La "cadena de archivos de recursos" incluye particularmente los archivos .rsr o . 4dr de bases de datos convertidas (abiertos automáticamente) y los archivos de recursos personalizados abiertos utilizando los comandos de este tema.*
- *Sin embargo, por razones relacionadas con la evolución de la arquitectura interna, ya no es posible acceder directamente a los recursos de la aplicación 4D ni a aquellos del sistema, bien sea a través de los comandos de este tema o utilizando referencias dinámicas. Algunos desarrolladores utilizan los recursos internos de 4D para sus interfaces (por ejemplo, los recursos que contienen los nombres de los meses o de los comandos del lenguaje). Esta práctica ahora está prohibida. En la mayoría de los casos, es posible utilizar otros medios en lugar de los recursos internos de 4D (constantes, comandos del lenguaje, etc.). Para limitar el impacto de esta modificación en bases existentes, se ha implementado un sistema de substitución, basado en externalizar los recursos que se utilizan con más frecuencia. Sin embargo, se recomienda cambiar las bases convertidas y eliminar las llamadas a recursos internos de 4D.*
- *Las bases de datos creadas con 4D a partir de la v11 no incluyen por defecto los archivos .RSR (recursos de estructura) y .4DR (recursos de datos).*

Principios actuales de gestión de recursos

En 4D v11, la noción de "recursos" ahora debe entenderse como "archivos necesarios para la traducción de la interfaz de las aplicaciones." La arquitectura actual de recursos está basada en una carpeta, llamada ***Resources****, que obligatoriamente debe estar ubicada junto al archivo de estructura de la base (.4db o .4dc). En esta carpeta, debe colocar todos los archivos que son necesarios para la traducción o personalización de la interfaz de la aplicación (archivos de tipo imagen, archivos de tipo texto, archivos XLIFF, etc.).*
También puede contener archivos de recursos de "antigua generación" de la base (archivos .rsr). Atención, estos archivos no se incluyen automáticamente en la cadena de recursos; deben abrirse utilizando los comandos estándar de gestión de recursos 4D. 4D utiliza mecanismos automáticos cuando trabaja con los contenidos de esta carpeta, particularmente para la gestión de archivos XLIFF (este punto se cubre en el Manual de Diseño). Por compatibilidad, los comandos ***Get indexed string*** *y* ***STRING LIST TO ARRAY*** *del tema "Recursos" permiten aprovechar esta arquitectura, sin embargo, ahora se recomienda utilizar el comando* ***Get localized string*** *del tema "Cadena de caracteres".*

CLOSE RESOURCE FILE

CLOSE RESOURCE FILE (resArchivo)

Parámetro	***Tipo***		***Descripción***
resArchivo	*DocRef*	➡	*Número de referencia del archivo de recursos*

Descripción

*El comando **CLOSE RESOURCE FILE** cierra el archivo de recursos cuyo número de referencia se pasa en resArchivo.*

*Incluso si ha abierto varias veces el mismo archivo de recursos, debe llamar **CLOSE RESOURCE FILE** sólo una vez para cerrarlo.*

*Si aplica **CLOSE RESOURCE FILE** al archivo de recursos de la aplicación 4D o de la base, el comando lo detecta y no hace nada.*

Si pasa un número de referencia de archivo de recursos incorrecto, el comando no hace nada.

*Recuerde llamar finalmente **CLOSE RESOURCE FILE** para un archivo de recursos que haya abierto utilizando **Open resource file**. Note que cuando sale de la aplicación (o abre otra base de datos), 4D cierra automáticamente todos los archivos de recursos abiertos.*

GET ICON RESOURCE

GET ICON RESOURCE (resNum ; resDatos {; resArchivo})

Parámetro	Tipo		Descripción
resNum	Entero largo	➡	Número de recurso icono
resDatos	Imagen	➡	Campo o variable imagen para recibir el icono
		⬅	Imagen resultante
resArchivo	DocRef	➡	Número de referencia de archivo de recursos o todos los archivos de recursos abiertos, si se omite

Nota de compatibilidad

Este comando no es soportado en las versiones de 64 bits de 4D. Si se ejecuta en este entorno se devuelve un error.

Descripción

*El comando **GET ICON RESOURCE** devuelve en el campo o la variable imagen resDatos, el icono guardado en el recurso icono ("cicn") cuyo número de identificación se pasa en resNum.*

Si no se encuentra el registro, el parámetro resDatos no cambia y la variable sistema OK toma el valor 0 (cero).

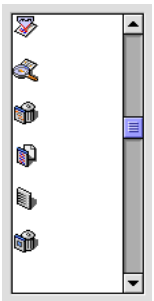
Si pasa un número de referencia de archivo de recursos válido en resArchivo, el recurso se busca en ese archivo únicamente. Si no pasa resArchivo, se devuelve la primera ocurrencia del recurso encontrada en la cadena de los archivos de recursos.

Ejemplo

El siguiente ejemplo carga en un array de tipo imagen los iconos color ubicados en la aplicación 4D activa:

```
If(On Windows)
    $vh4DResArchivo:=Open resource file(Replace string(Application file;".EXE";".RSR"))
Else
    $vh4DResArchivo:=Open resource file(Application file)
End if
RESOURCE LIST("cicn";$alResNum;$asResNom;$vh4DResArchivo)
$vlNbIcons:=Size of array($alResNum)
ARRAY PICTURE(ag4DIcon;$vlNbIconos)
For($vlElem;1;$vlNbIconos)
    GET ICON RESOURCE($alResNum{$vlElem};ag4DIcon{$vlElem};$vh4DResArchivo)
End for
```

Una vez ejecutado este código, el array se verá de esta forma cuando se muestre en un formulario:



Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si se encuentra el recursos, de lo contrario toma el valor 0 (cero).

Get indexed string

Get indexed string (resNum ; strNum {; resArchivo}) -> Resultado

Parámetro	Tipo		Descripción
resNum	Entero largo	➡	Número de recurso o Atributo "id" del elemento "grupo" (XLIFF)
strNum	Entero largo	➡	Número de cadena o Atributo "id" del elemento "trans-unit" (XLIFF)
resArchivo	DocRef	➡	Número de referencia del archivo de recursos Si se omite: todos los archivos XLIFF o los archivos de recursos abiertos
Resultado	Cadena	↩	Valor de la cadena indexada

Descripción

*El comando **Get indexed string** devuelve:*

- *Una de las cadenas guardadas en el recurso lista de cadenas ("STR#") cuyo número de identificación se pasa en resNum.*
- *Una cadena guardada en un archivo XLIFF abierto cuyo atributo "id" del elemento "grupo" se pasa en resNum (ver a continuación "Compatibilidad con la arquitectura XLIFF). Pase el número de la cadena en strNum. Las cadenas de un recurso lista de cadenas están numeradas de 1 a N. Para recuperar todas las cadenas (y sus números) de un recurso lista de cadenas, utilice el comando **STRING LIST TO ARRAY**. Si el recurso no se encuentra, o si la cadena no está al interior del recurso, una cadena vacía se devuelve y la variable sistema OK toma el valor 0 (cero). Si pasa un número de referencia de archivo de recursos válido en resArchivo, el recurso se busca en ese archivo únicamente. Si no pasa resArchivo, se devuelve la primera ocurrencia del recurso encontrada en la cadena de archivos de recursos.*

***Nota:** una cadena de un recurso lista de cadenas puede contener hasta 255 caracteres.*

Compatibilidad con la arquitectura XLIFF

*El comando **Get indexed string** es compatible con la arquitectura XLIFF de 4D a partir de la versión 11: el comando busca primero por los valores correspondientes a resNum y strNum en todos los archivos XLIFF abiertos (si el parámetro resArchivo se omite). En este caso, resNum especifica el atributo **id** del elemento **grupo y** strNum especifica el atributo **id** del elemento **trans-unit**. Si no se encuentra el valor, el comando continua la búsqueda en los archivos de recursos abiertos. Para mayor información sobre la arquitectura XLIFF en 4D, consulte el Manual de Diseño.*

Variables y conjuntos del sistema

Si el recurso se encuentra, OK toma el valor 1, de lo contrario toma el valor 0 (cero).

GET PICTURE RESOURCE

GET PICTURE RESOURCE (resNum ; resDatos {; resArchivo})

Parámetro	Tipo		Descripción
resNum	Entero largo	➡	Número de recurso
resDatos	Campo, Variable	➡	Campo o variable imagen a recibir la imagen
		⬅	Contenido del recurso PICT
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o todos los archivos de recursos abiertos, si se omite

Descripción

*El comando **GET PICTURE RESOURCE** devuelve en el campo o en la variable resDatos la imagen guardada en el recurso imagen ("PICT") cuyo número se pasa en resNum.*

Si el recurso no se encuentra, el parámetro resDatos no se modifica y la variable OK toma el valor 0 (cero).

Si pasa un número de referencia de archivo de recursos válido en resArchivo, el recurso se busca en ese archivo únicamente. Si no pasa resArchivo, se devuelve la primera ocurrencia del recurso encontrado en la cadena de archivos de recursos.

***Nota:** el tamaño de un recurso imagen puede ser de varios megabytes.*

Ejemplo

*Ver el ejemplo del comando **RESOURCE LIST**.*

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si se encuentra el recurso, de lo contrario toma el valor 0 (cero).

Gestión de errores

*Si no hay suficiente memoria para cargar la imagen, se genera un error. Puede interceptar este error con la ayuda de un método de gestión de errores instalado por el comando **ON ERR CALL**.*

GET RESOURCE

GET RESOURCE (resTipo ; resNum ; resDatos {; resArchivo})

Parámetro	Tipo		Descripción
resTipo	Cadena	➡	Tipo de recurso (4 caracteres)
resNum	Entero largo	➡	Número de recurso
resDatos	BLOB	➡	Campo o variable BLOB a recibir los datos
		⬅	Contenido del recurso
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o todos los archivos de recursos abiertos, si se omite

Descripción

*El comando **GET RESOURCE** devuelve en el campo o la variable BLOB resDatos el contenido del recurso cuyo tipo y número se pasa en resTipo y resNum.*

***Importante:** debe pasar una cadena de 4 caracteres en resTipo.*

Si no se encuentra el recurso, el parámetro resDatos no cambia y la variable OK toma el valor 0 (cero).

Si pasa un número de referencia de archivo de recursos válido en resArchivo, el recurso se busca en ese archivo únicamente. Si no pasa resArchivo, se devuelve la primera ocurrencia del recurso encontrada en la cadena de archivos de recursos.

***Nota:** el tamaño de un recurso puede ser de varios megabytes.*

Independencia de plataforma

*Recuerde que trabaja con recursos basados en Mac OS. Sin importar la plataforma, los valores internos de los recursos como los Enteros largos son almacenados utilizando ordenación de bytes ("byte ordering") Macintosh. En Windows, para los datos de los recursos estándar (tales como los recursos listas de cadenas y los recursos imágenes) la ordenación de bytes es automáticamente inversa ("byte swapping") cuando es necesario. Por otra parte, si crea y utiliza sus propias estructuras de datos internas, usted decide si aplica la ordenación inversa a los datos que extrajo del BLOB (por ejemplo, al pasar **Macintosh byte ordering** a un comando como **BLOB to longint**).*

Ejemplo

*Ver el ejemplo del comando **SET RESOURCE**.*

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si se encuentra el recurso, de lo contrario toma el valor 0 (cero).

Gestión de errores

*Si no hay suficiente memoria para cargar la imagen, se genera un error. Puede interceptar este error con la ayuda de un método de gestión de errores instalado por el comando **ON ERR CALL**.*

Get resource name

Get resource name (resTipo ; resNum {; resArchivo}) -> Resultado

Parámetro	Tipo		Descripción
resTipo	Cadena	→	Tipo de recurso (4 caracteres)
resNum	Entero largo	→	Número de referencia del recurso
resArchivo	DocRef	→	Número de referencia del archivo de recursos o Todos los archivos de recursos abiertos, si se omite
Resultado	Cadena	↩	Nombre del recurso

Descripción

*El comando **Get resource name** devuelve el nombre del recurso cuyo tipo se pasa en resTipo y cuyo número de referencia (ID) en resNum.*

Si pasa un número de referencia de archivo de recursos en el parámetro resArchivo, el recurso se busca en ese archivo únicamente. Si no pasa resArchivo, el archivo se busca en los archivos de recursos abiertos.

*Si el recurso no existe, **Get resource name** devuelve una cadena vacía.*

Get resource properties

Get resource properties (resTipo ; resNum {; resArchivo}) -> Resultado

Parámetro	Tipo		Descripción
resTipo	Cadena	➡	Tipo de recurso (4 caracteres)
resNum	Entero largo	➡	Número de referencia del recurso (ID)
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o Todos los archivos de recursos abiertos, si se omite
Resultado	Entero largo	↩	Atributos del recurso

Descripción

*El comando **Get resource properties** devuelve los atributos del recurso cuyo tipo se pasa en resTipo y cuyo número de identificación se pasa en resNum.*

Si pasa un número de referencia de archivo de recursos válido en resArchivo, el recurso se busca en ese archivo únicamente. Si no pasa resArchivo, el archivo se busca en los archivos de recursos abiertos.

*Si el archivo de recursos no existe, **Get resource properties** devuelve 0 (cero) y la variable OK toma el valor 0 (cero).*

*El valor numérico devuelto por **Get resource properties** debe considerable como un valor binario cuyos bits tienen un significado especial.*

Ejemplo

*Ver el ejemplo del comando **Get resource name**.*

Variables y conjuntos del sistema

La variable sistema OK toma el valor 0 si el recurso no existe, de lo contrario toma el valor 1.

Get string resource

Get string resource (resNum {; resArchivo}) -> Resultado

Parámetro	Tipo		Descripción
resNum	Entero largo	➡	Número del recurso
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o Todos los archivos de recursos abiertos, si se omite
Resultado	Cadena	↩	Contenido del recurso STR

Descripción

*El comando **Get string resource** devuelve la cadena almacenada en el recurso cadena ("STR ") cuyo número de referencia se pasa en resNum.*

Si el recurso no se encuentra, se devuelve una cadena vacía y la variable OK toma el valor 0 (cero).

Si pasa un número de referencia de archivo de recursos válido en resArchivo, el recurso se busca en ese archivo únicamente. Si no pasa resArchivo, se devuelve la primera ocurrencia del recurso encontrada en la cadena de archivos de recursos.

***Nota:** un recurso cadena puede contener hasta 255 caracteres.*

Ejemplo

El siguiente ejemplo muestra los contenidos del recurso cadena de ID=20911, que debe encontrar en al menos uno de los archivos de recursos abiertos:

```
ALERT(Get string resource(20911))
```

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si se encuentra el recurso, de lo contrario toma el valor 0 (cero).

Get text resource

Get text resource (resNum {; resArchivo}) -> Resultado

Parámetro	Tipo		Descripción
resNum	Entero largo	➡	Número de recurso
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o todos los archivos de recursos abiertos, si se omite
Resultado	Texto	↩	Contenido del recurso TEXT

Descripción

*El comando **Get text resource** devuelve el texto guardado en el recurso texto ("TEXT") cuyo número de identificación se pasa en resNum.*

Si no se encuentra el recurso, se devuelve un texto vacío y la variable sistema OK toma el valor 0 (cero).

Si pasa un número de referencia de archivo de recursos válido en resArchivo, el recurso se busca en ese archivo únicamente. Si no pasa resArchivo, será devuelta la primera ocurrencia del recurso encontrado en la cadena de archivos de recursos.

***Nota:** un recurso texto puede contener hasta 32 000 caracteres.*

Ejemplo

El siguiente ejemplo muestra el contenido del recurso texto de ID=20800, que debe estar ubicado en al menos uno de los archivos de recursos abiertos:

```
ALERT(Get text resource(20800))
```

Variables y conjuntos del sistema

Si se encuentra el recurso, OK toma el valor 1. De lo contrario, toma el valor 0 (cero).

Open resource file

Open resource file (resNomArchivo {; tipo}) -> Resultado

Parámetro	Tipo		Descripción
resNomArchivo	Cadena	➡	Nombre o ruta de acceso completa del archivo de recursos o Cadena vacía para mostrar la caja de diálogo estándar de apertura de archivos
tipo	Cadena	➡	Tipo de archivo Mac OS (cadena de 4 caracteres) o extensión del archivo Windows (cadena de 1 a 3 caracteres) o archivo de recursos ("res " / .RES) si se omite
Resultado	DocRef	⬅	Número de referencia del archivo de recursos

Descripción

El comando ***Open resource file*** *abre el archivo de recursos cuyo nombre o ruta de acceso completa se pasa en resNomArchivo.*

Si pasa un nombre de archivo, el archivo debe estar ubicado en la misma carpeta que el archivo de estructura de la base. Para abrir un archivo de recursos ubicado en otra carpeta, pase una ruta de acceso completa.

Si pasa una cadena vacía en resNomArchivo, la caja de diálogo estándar de apertura de archivos aparece, permitiendo al usuario seleccionar el archivo a abrir. Si el usuario hace clic en Cancelar en esta caja de diálogo, no se abre ningún archivo de recursos; ***Open resource file*** *devuelve un valor nulo en DocRef y la variable OK toma el valor 0.*

Si el archivo de recursos se abre correctamente, ***Open resource file*** *devuelve su número de referencia de archivo y la variable OK toma el valor 1. Si el archivo de recursos no existe o si el archivo que intenta abrir no es un archivo de recursos, se genera un error.*

- *En Macintosh, si utiliza la caja de diálogo estándar de apertura de archivos, todos los archivos se presentan por defecto. Para mostrar archivos de un tipo en particular, especifique el tipo del archivo en el parámetro opcional tipoArchivo.*
- *En Windows, si utiliza la caja de diálogo estándar de apertura de archivos, todos los archivos se presentan por defecto. Para mostrar archivos de un tipo particular, pase en tipoArchivo, una extensión de archivo Windows de 1 a 3 caracteres o un tipo de archivo Macintosh asociado a una extensión Windows utilizando el comando* ***_o_MAP FILE TYPES****.*

Recuerde llamar ***CLOSE RESOURCE FILE*** *para el archivo de recursos. Note, sin embargo, que 4D cierra automáticamente todos los archivos de recursos abiertos utilizando* ***Open resource file*** *cuando sale de la aplicación o abre otra base de datos.*

A diferencia del comando ***Open document****, que abre por defecto un documento con un acceso exclusivo en lectura escritura,* ***Open resource file*** *permite abrir un archivo de recursos ya abierto desde la sesión 4D. Por ejemplo, si trata de abrir el mismo documento dos veces con* ***Open document****, un error de E/S será devuelto al segundo intento. Por otra parte, si trata de abrir un archivo de recursos ya abierto desde la sesión 4D,* ***Open resource file*** *devolverá su número de referencia. Incluso si abre un archivo de recursos varias veces, sólo necesita llamar* ***CLOSE RESOURCE FILE*** *una vez para cerrar ese archivo. Note que este funcionamiento es válido sólo si el archivo de recursos está abierto desde la sesión 4D; si trata de abrir un archivo de recursos ya abierto por otra aplicación, obtendrá un error E/S.*

Advertencia:

- *Está prohibido acceder a los archivos de recursos de las aplicaciones 4D y de las bases fusionadas con 4D Desktop.*
- *Aunque es técnicamente posible, no es recomendable utilizar el archivo de recursos de la estructura de la base porque su código no funcionará si la base está compilada y fusionada con 4D Desktop.*
  Sin embargo, si accede al archivo de recursos de la estructura y quiere añadir, borrar o modificar los recursos por programación, asegúrese de probar el entorno en el cual se ejecuta la base. Con 4D Server, esto probablemente llevará a serios problemas. Por ejemplo, si modifica un recurso en el equipo del servidor (vía un método de base de datos o un procedimiento almacenado), definitivamente afectará el sistema de administración integrado de 4D Server que distribuye recursos (de manera transparente) a las estaciones de trabajo. Note que con 4D Client, usted no tiene acceso directo al archivo de estructura; está ubicado en el equipo servidor.
- *Por todas estas razones, si utiliza los recursos, guárdelos en sus propios archivos.*
- *Cuando trabaje con sus propios recursos, NO utilice números de recursos negativos; los números negativos están reservados para el sistema operativo. NO utilice números de recursos entre 0 y 14 999; este rango está reservado para 4D. Utilice el rango entre 15 000 y 32 767 para sus propios recursos. Recuerde que una vez haya abierto un archivo de recursos, será el primer archivo donde se buscará en la cadena de archivos de recursos. Si guarda un recurso en ese archivo con un número que pertenece a los rangos reservados para el sistema o para 4D, este recurso será utilizado no sólo por comandos como* ***GET RESOURCE*** *si no también por rutinas internas de la aplicación 4D. Este puede podría ser el resultado que usted quiere obtener, pero si no está seguro, NO utilice estos rangos, ya que pueden producir errores del sistema.*
- *Un archivo de recursos es altamente estructurado y no puede aceptar más de 2 700 recursos por archivo. Si quiere trabajar con archivos que contengan un gran número de recursos, es recomendable probar ese número antes de añadir nuevos recursos a un archivo. Consulte los ejemplos de* ***Count resources*** *en la descripción del comando* ***RESOURCE TYPE LIST****.*

Una vez haya abierto una archivo de recursos, puede analizar su contenido utilizando los comandos ***RESOURCE TYPE LIST*** *y* ***RESOURCE LIST****.*

Ejemplo 1

El siguiente ejemplo trata de abrir, en Windows, el archivo de recursos "MyPrefs.res" ubicado en la carpeta de la base:

```
$vhResArchivo:=Open resource file("MisPrefs";"res ")
```

En Macintosh, el ejemplo trata de abrir el archivo "MisPrefs".

Ejemplo 2

El siguiente ejemplo trata de abrir en Windows el archivo de recursos "MisPrefs.rsr" ubicado en la carpeta de la base:

```
$vhResArchivo:=Open resource file("MisPrefs";"rsr")
```

En Mac OS, el ejemplo tratará de abrir el archivo "MisPrefs".

Ejemplo 3

El siguiente ejemplo muestra la caja de diálogo estándar de apertura de archivos, en la cual se muestran todos los tipos de documentos:

```
$vhResArchivo:=Open resource file("")
```

Ejemplo 4

El siguiente ejemplo muestra la caja de diálogo estándar de apertura de archivos, en la cual sólo se muestran los documentos creados con la ayuda de la función ***Create resource file****, y que utilizan el tipo por defecto:*

```
$vhResArchivo:=Open resource file("";"res ")
If(OK=1)
   ALERT("Acaba de abrir""+Document+"".")
   CLOSE RESOURCE FILE($vhResArchivo)
End if
```

Variables y conjuntos del sistema

Si el archivo de recursos se abre correctamente, la variable sistema OK toma el valor 1. Si el archivo de recursos no se pudo abrir o si el usuario hace clic en Cancelar en la caja de diálogo estándar de apertura de archivos, la variable OK toma el valor 0 (cero).

Si el archivo de recursos se abre correctamente utilizando la caja de diálogo estándar de apertura de archivos, la variable sistema Document contiene la ruta de acceso al archivo.

Gestión de errores

Si el archivo de recursos no se pudo abrir por un problema del recurso o de E/S, se genera un error. Puede interceptar este error con un método de gestión de errores instalado por el comando ***ON ERR CALL****.*

RESOURCE LIST

RESOURCE LIST (resTipo ; resNums ; resNoms {; resArchivo})

Parámetro	Tipo		Descripción
resTipo	Cadena	➡	Tipo de recurso (4 caracteres)
resNums	Array entero largo	⬅	Números de recursos de este tipo
resNoms	Array cadena	⬅	Nombres de los recursos de este tipo
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o Todos los archivos de recursos abiertos, si se omite

Descripción

*El comando **RESOURCE LIST** llena los arrays resNums y resNoms con los números y los nombres de los recursos cuyo tipo se pasa en resTipo.*

***Importante:** debe pasar una cadena de 4 caracteres en resTipo.*

Si pasa un número de referencia de archivo de recursos válido en el parámetro opcional resArchivo, sólo se listan los recursos presentes en este archivo. Si no pasa resArchivo, se listan todos los recursos de los archivos de recursos abiertos.

*Si predeclara los arrays antes de llamar a **RESOURCE LIST**, debe predeclarar resNums como un array de tipo Entero largo y resNoms como un array de tipo Alfa o Texto. Si no predecla los arrays, el comando crea resNums como un array de tipo Entero largo y resNoms como un array de tipo Texto.*

*Después de la llamada, puede probar el número de recursos encontrados aplicando el comando **Size of Array** al array resNums o resNoms.*

Ejemplo 1

El siguiente ejemplo llena los arrays $alResNum y $atResNom con los números y los nombres de recursos de tipo lista de cadenas presentes en el archivo de estructura de la base:

```
If (On Windows)
   $vhEstructuraResArchivo:=Open resource file(Replace string(Structure file;".4DB";".RSR"))
Else
   $vhEstructuraResArchivo:=Open resource file(Structure file)
End if
If (OK=1)
   RESOURCE LIST("STR#";$alResNum;$atResNom;$vhEstructuraResArchivo)
End if
```

Ejemplo 2

El siguiente ejemplo copia los recursos imagen presentes en todos los archivos de recursos abiertos en la librería de imágenes de la base:

```
RESOURCE LIST("PICT";$alResNum;$atResNom)
Open window(50;50;550;120;5;"Copia de los recursos PICT...")
For($vlElem;1;Size of array($alResNum))
   GET PICTURE RESOURCE($alResNum{$vlElem};$vgImagen)
   If(OK=1)
      $vsNombre:=$atResNom{$vlElem}
      If($vsNombre="")
         $vsNombre:="PICT resNum="+String($alResNum{$vlElem})
      End if
      ERASE WINDOW
      GOTO XY(2;1)
      MESSAGE("Añada la imagen ""+$vsNombre+""a la librería de imágenes de la base.")
      SET PICTURE TO LIBRARY($vgImagen;$alResNum{$vlElem};$vsNombre)
   End if
End for
CLOSE WINDOW
```

RESOURCE TYPE LIST

RESOURCE TYPE LIST (resTipos {; resArchivo})

Parámetro	Tipo		Descripción
resTipos	Array cadena	⬅	Lista de tipos de recursos disponibles
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o Todos los archivos de recursos abiertos, si se omite

Descripción

*El comando **RESOURCE TYPE LIST** llena el array resTipos con los tipos de recursos presentes en el (los) archivo(s) de recursos abierto(s).*

Si pasa un número de referencia de archivo de recursos válido en el parámetro opcional resArchivo, sólo se listan los recursos de ese archivo. Si no pasa resArchivo, se listan todos los recursos de los archivos de recursos abiertos.

*Puede predeclarar el array resTipos como un array tipo Alfa o Texto antes de llamar a **RESOURCE TYPE LIST**. Si no predeclara el array, el comando crea resTipos como un array tipo Texto.*

*Después de la llamada, puede probar el número de tipo de recursos encontrados aplicando el comando **Size of Array** al array resTipos.*

Ejemplo 1

El siguiente ejemplo llena el array atResTipo con los tipos de recursos presentes en los archivos de recursos abiertos:

```
RESOURCE TYPE LIST(atResTipo)
```

Ejemplo 2

El siguiente archivo muestra si el archivo de estructura Mac OS utiliza el contenido de los antiguos plug-ins 4D, que deberán ser actualizados para utilizar la base en Windows:

```
$vhResArchivo:=Open resource file(Structure file)
RESOURCE TYPE LIST(atResTipo;$vhResArchivo)
If(Find in array(atResTipo;"4DEX")>0)
   ALERT("Esta base contiene los plug-ins 4D del modelo antiguo."+(Char(13)*2)+
    "Tendrá que actualizarlos para utilizar esta base en Windows.")
End if
```

***Nota:** el archivo de estructura no es el único archivo en el cual los plug-ins de la versión anterior pueden ser almacenados. La base también puede incluir un archivo Proc.Ext.*

Ejemplo 3

El siguiente método de proyecto devuelve el número de recursos presentes en un archivo de recursos:

```
` Método de proyecto Contar recursos
` Contar recursos ( Hora) -> Entero largo
` Contar recursos ( DocRef ) -> Número de recursos

C_LONGINT($0)
C_TIME($1)

$0:=0
RESOURCE TYPE LIST($atResTipo;$1)
For($vlElem;1;Size of array($atResTipo))
   RESOURCE LIST($atResTipo{$vlElem};$alResNum;$atResNom;$1)
   $0:=$0+Size of array($alResNum)
End for
```

Una vez este método de proyecto se implementa en una base, puede escribir:

```
$vhResArchivo:=Open resource file("")
If(OK=1)
   ALERT("El archivo ""+Document+"" contiene "+String(Count resources($vhResArchivo))+" recurso(s).")
   CLOSE RESOURCE FILE($vhResArchivo)
End if
```

STRING LIST TO ARRAY

STRING LIST TO ARRAY (resNum ; cadenas {; resArchivo})

Parámetro	Tipo		Descripción
resNum	Entero largo	➡	Número de referencia del recurso o Atributo "id" del elemento "group" (XLIFF)
cadenas	Array cadena	⬅	Cadenas del elemento "group" (XLIFF)
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o Todos los archivos XLIFF o los archivos de recursos abiertos, si se omite

Descripción

*El comando **STRING LIST TO ARRAY** llena el array cadenas con:*

- *las cadenas almacenadas en el recurso de tipo lista de cadenas ("STR#") cuyo número se pasa en resNum.*
- *o con una cadena almacenada en un archivo XLIFF abierto cuyo atributo "id" del elemento "group" se pasa en resNum (ver a continuación "Compatibilidad con la arquitectura XLIFF").*

Si el recurso no se encuentra, el array cadenas no cambia y la variable OK toma el valor 0 (cero).
*Si pasa un número de referencia de archivo de recursos válido en resArchivo, el recurso se busca en ese archivo únicamente. Si no pasa resArchivo, se devolverá la primera ocurrencia del recurso encontrado en la cadena de archivos de recursos. Antes de llamar **STRING LIST TO ARRAY**, puede predeclarar el array cadenas como una array de tipo cadena o texto. Si no predeclara el array, el comando crea cadenas como un array de tipo Texto.*

***Nota:** cada cadena de un recurso lista de cadenas puede contener hasta 255 caracteres.*

***Tip:** cuando utilice los recursos listas de cadenas, limítese a recursos de 32K, y a un máximo de unas centenas de cadenas por recurso.*

Compatibilidad con la arquitectura XLIFF

*El comando **STRING LIST TO ARRAY** es compatible con la arquitectura XLIFF de 4D a partir de la versión 11: el comando busca primero por los valores correspondientes a resNum y strNum en todos los archivos XLIFF abiertos (si el parámetro resArchivo se omite) y llena el array cadenas con los valores correspondientes. En este caso, resNum específica el atributo **id** del elemento **group** y el array cadenas contiene todas las cadenas del elemento. Si no se encuentra el valor, el comando continua la búsqueda en los archivos de recursos abiertos. Para mayor información sobre la arquitectura XLIFF en 4D, consulte el Manual de Diseño.*

Variables y conjuntos del sistema

Si se encuentra el recurso, la variable sistema OK toma el valor 1, de lo contrario toma el valor 0 (cero).

_o_ARRAY TO STRING LIST

_o_ARRAY TO STRING LIST (cadenas ; resNum {; resArchivo})

Parámetro	Tipo		Descripción
cadenas	Array cadena	➡	Array alfa o texto (nuevo contenido del recurso STR#)
resNum	Entero largo	➡	Número de recurso
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o archivo de recursos actual, si se omite

Descripción

Nota de compatibilidad: *los comandos que escriben recursos ya no funcionan y no deben utilizarse.*

_o_Create resource file

_o_Create resource file (resNomArchivo {; tipo {; *}}) -> Resultado

Parámetro	Tipo		Descripción
resNomArchivo	Cadena	➡	Nombre o ruta de acceso completa del archivo de recursos o cadena vacía para mostrar la caja de diálogo para guardar archivos.
tipo	Cadena	➡	Tipo de archivo Mac OS (cadena de 4 caracteres) o extensión del archivo Windows (cadena de 1 a 3 caracteres) o archivo de recursos ("res " / .RES), si se omite
*		➡	Si se pasa = Utilizar el data fork
Resultado	DocRef	⬅	Número de referencia del archivo de recursos

Descripción

Nota: *los comandos que escriben recursos ya no funcionan y no deben utilizarse.*

_o_DELETE RESOURCE

_o_DELETE RESOURCE (resTipo ; resNum {; resArchivo})

Parámetro	Tipo		Descripción
resTipo	Cadena	➡	Tipo de recurso (4 caracteres)
resNum	Entero largo	➡	Número de recurso
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o archivo de recursos actual, si se omite

Descripción

Desde 4D v13, los comandos que permiten escribir en los archivos de recursos son obsoletos y ya no deben utilizarse.

_o_Get component resource ID

_o_Get component resource ID (nomComp ; resTipo ; resNumOriginal) -> Resultado

Parámetro	Tipo		Descripción
nomComp	Cadena	→	Nombre del componente que hace referencia al recurso
resTipo	Cadena	→	Tipo de recurso (4 caracteres), PICT o STR#
resNumOriginal	Entero largo	→	Número original del recurso antes de la instalación del componente
Resultado	Entero largo	←	Número actual del recurso

Descripción

Nota de compatibilidad: *este comando funcionaba con los componentes de antigua generación, incompatibles con 4D versión 11 SQL y posteriores. No tiene ningún efecto y no debe utilizarse más.*

_o_SET PICTURE RESOURCE

_o_SET PICTURE RESOURCE (resNum ; resDatos {; resArchivo})

Parámetro	Tipo		Descripción
resNum	Entero largo	➡	Número del recurso
resDatos	Imagen	➡	Nuevo contenido del recurso PICT
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o archivo de recursos actual, si se omite

Descripción

*N**ota de compatibilidad:** los comandos que permiten escribir en los archivos de recursos ya no funcionan y no deben utilizarse.*

_o_SET RESOURCE

_o_SET RESOURCE (resTipo ; resNum ; resDatos {; resArchivo})

Parámetro	Tipo		Descripción
resTipo	Cadena	➡	Tipo de recurso (4 caracteres)
resNum	Entero largo	➡	Número de recurso
resDatos	BLOB	➡	Nuevo contenido del recurso
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o Archivo de recursos actual, si se omite

Descripción

*N**ota de compatibilidad:** los comandos que permiten escribir en los archivos de recursos ya no funcionan y no deben utilizarse.*

_o_SET RESOURCE NAME

_o_SET RESOURCE NAME (resTipo ; resNum ; resNombre {; resArchivo})

Parámetro	Tipo		Descripción
resTipo	Cadena	➡	Tipo de recurso (4 caracteres)
resNum	Entero largo	➡	Número de referencia del recurso (ID)
resNombre	Cadena	➡	Nuevo nombre del recurso
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o Todos los archivos de recursos abiertos, si se omite

Descripción

Nota: *los comandos que escriben recursos ya no funcionan y no deben utilizarse.*

_o_SET RESOURCE PROPERTIES

_o_SET RESOURCE PROPERTIES (resTipo ; resNum ; resAttr {; resArchivo})

Parámetro	Tipo		Descripción
resTipo	Cadena	➡	Tipo de recurso (4 caracteres)
resNum	Entero largo	➡	Número de referencia del recurso
resAttr	Entero largo	➡	Nuevos atributos del recurso
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o Archivo de recursos actual, si se omite

Descripción

*N**ota de compatibilidad:** los comandos que permiten escribir en los archivos de recursos ya no funcionan y no deben utilizarse.*

_o_SET STRING RESOURCE

_o_SET STRING RESOURCE (resNum ; resDatos {; resArchivo})

Parámetro	Tipo		Descripción
resNum	Entero largo	➡	Número de recurso
resDatos	Cadena	➡	Nuevo contenido del recurso STR
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o archivo de recursos actual, si se omite

Descripción

Nota de compatibilidad: *los comandos que escriben recursos ya no funcionan y no deben utilizarse.*

_o_SET TEXT RESOURCE

_o_SET TEXT RESOURCE (resNum ; resDatos {; resArchivo})

Parámetro	Tipo		Descripción
resNum	Entero largo	➡	Número de recurso
resDatos	Cadena	➡	Nuevo contenido del recurso TEXT
resArchivo	DocRef	➡	Número de referencia del archivo de recursos o Archivo de recursos actual, si se omite

Descripción

Nota de compatibilidad: *los comandos que permiten escribir en los archivos de recursos ya no funcionan y no deben utilizarse.*

Registros

- *Acerca de números de registros*
- *Uso de la pila de registros*
- *CREATE RECORD*
- *DELETE RECORD*
- *DISPLAY RECORD*
- *DUPLICATE RECORD*
- *GOTO RECORD*
- *Is new record*
- *Is record loaded*
- *Modified record*
- *POP RECORD*
- *PUSH RECORD*
- *Record number*
- *Records in table*
- *SAVE RECORD*
- *Sequence number*

Acerca de números de registros

En 4D, tres números están asociados a un registro:

- *Número de registro*
- *Número en la selección*
- *Número automático*

Número de registro

El número de registro es el número físico/absoluto del registro. Este número se registra automáticamente para cada nuevo registro y permanece constante hasta que se borra el registro. Los números de registro comienzan desde cero (0).

Los números de registro no son únicos porque los números de registros borrados son reutilizados para los nuevos registros. Igualmente estos números son modificados cuando la base se repara o compacta.

Número en la selección

El número en la selección es la posición del registro en la selección actual. Este número depende de la selección actual. Si la selección se modifica u ordena, este número probablemente también cambiará. La numeración en una selección actual comienza en uno (1).

Número automático

*El número automático es un número único, no repetible, que puede ser asignado a un campo de un registro (vía la propiedad **Autoincrementar**, el atributo SQL **AUTO_INCREMENT** o el comando **Sequence number**). Este número no se almacena automáticamente con cada registro. Comienza por defecto en 1 y se incrementa cada vez que se crea un nuevo registro. A diferencia de los números de registros, un número automático no se reutiliza cuando se borra un registro o cuando la base se compacta o repara.*

Los números de secuencia ofrecen una forma de tener un número de identificación único para cada registro. Si un número automático se incrementa durante una transacción, el número no se reduce si la transacción se cancela.

***Nota:** 4D no realiza ninguna verificación al modificar el contador interno automático de una tabla utilizando el comando **SET DATABASE PARAMETER**. Si reduce este contador, los nuevos registros creados podrían tener números que ya han sido asignados.*

Ejemplos de números de registros

Las siguientes tablas ilustran los números que están asociados con registros. Cada línea de la tabla representa la información sobre un registro. El orden de las líneas es el orden en el cual los registros serán mostrados en un formulario de salida.

- ***Columna de datos**: los valores de un campo en cada registro. En nuestro ejemplo, contiene un nombre de una persona.*
- ***Columna de número de registro**: este es el número absoluto del registro, que es devuelto por la función **Record Number**.*
- ***Columna del número en la selección**: este es el número de posición en la selección actual, que es devuelto por la función **Selected record Number**.*
- ***Columna del número automático**: este es el número único del registro, que es devuelto por la función **Sequence Number**. Este número se almacena en un campo.*

Después de la entrada de registros

La primera tabla muestra los registros después de ser introducidos.

• El orden de los registros por defecto es por el número de registro.
• El número de registro comienza por 0.
• El número en la selección y el número automático comienzan por 1.

Datos	Número registro	Número selección	Número automático
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Sam	3	4	4
Lisa	4	5	5

***Nota**: los registros permanecen en el orden por defecto después de que un comando modifica la selección actual sin reordenarla, como por ejemplo el comando de menú Mostrar todos en el entorno Diseño o después de la ejecución del comando **ALL RECORDS**.*

Después de ordenar los registros

La siguiente tabla muestra los mismos registros ordenados por nombre.

• El número de registro permanece asociado con cada registro.
• El número en la selección refleja la nueva posición del registro en la selección ordenada.
• El número automático nunca cambia, ya que se asigna cuando se crea cada registro y se almacena con él.

Datos	Número registro	Número selección	Número automático
Lisa	4	1	5
Sabra	2	2	3
Sam	3	3	4
Terri	1	4	2
Tess	0	5	1

Después la eliminación de un registro

La siguiente tabla muestra los registros después de borrar el registro de Sam.

- Sólo los números en la selección han cambiado. Los números en la selección reflejan el orden de visualización de los registros.

Datos	Número registro	Número selección	Número automático
Lisa	4	1	5
Sabra	2	2	3
Terri	1	3	2
Tess	0	4	1

Después de añadir un registro

La siguiente tabla muestra los registros después de añadir el registro Liz.

- Un nuevo registro se añade al final de la selección actual.
- El número de registro de Sam se utiliza nuevamente para el nuevo registro.
- El número automático se incrementa en 1.

Datos	Número registro	Número selección	Número automático
Tess	0	1	1
Terri	1	2	2
Sabra	2	3	3
Lisa	4	4	5
Liz	3	5	6

Después de un cambio de selección y una ordenación

La siguiente tabla muestra los registros después de que la selección se redujo a tres registros y luego se ordenó.

- Sólo cambia el número en la selección.

Datos	Número registro	Número selección	Número automático
Sabra	2	1	3
Liz	3	2	6
Terri	1	3	2

Uso de la pila de registros

---

*Los comandos **PUSH RECORD** y **POP RECORD** permiten colocar ("apilar") registros en la pila de registros, y eliminar ("desapilar") de la pila.*

Cada proceso dispone de su propia pila de registros para cada tabla. 4D administra las pilas de registros por usted. Cada pila de registros es de tipo LIFO (último en entrar, primero en salir). La capacidad de la pila depende de la memoria.

***PUSH RECORD** y **POP RECORD** deben ser utilizados con prudencia. Cada registro que es apilado utiliza una parte de la memoria disponible. Apilar demasiados registros puede causar la aparición de un mensaje del tipo "memoria insuficiente" o una pila llena.*

4D quita de la pila de registros al regresar al menú al final de la ejecución del método.

***PUSH RECORD** y **POP RECORD** son útiles cuando quiera examinar registros que se encuentran en la misma tabla que está utilizando. Para hacer esto, usted apila el registro, busca y examina los registros en la tabla (copia campos en variables, por ejemplo), y finalmente desapila el registro para restaurar el registro.*

***Nota para los usuarios de la versión 3**: si cuando introduce un registro, debe verificar la unicidad de un valor en varios campos, utilice el comando **SET QUERY DESTINATION**. Esto le evitará realizar las llamadas a **PUSH RECORD** y **POP RECORD** que estaba haciendo antes y después de la llamada a **QUERY** para preservar los datos introducidos en el registro actual. **SET QUERY DESTINATION** le permite ejecutar una búsqueda que no cambia la selección ni los registros actuales.*

CREATE RECORD

CREATE RECORD {(tabla)}

Parámetro	Tipo	Descripción
tabla	Tabla ➡	Tabla para la cual crear un nuevo registro, o Tabla por defecto, si se omite

Descripción

CREATE RECORD crea un nuevo registro vacío para tabla, pero no muestra el nuevo registro. Utilice el comando **ADD RECORD** para crear un nuevo registro y mostrarlo en un formulario de entrada.

CREATE RECORD se utiliza en lugar de **ADD RECORD** cuando los valores de los registros se introducen por programación. El nuevo registro se convierte en el registro actual para la selección actual pero la selección actual no se modifica.

El registro existe en memoria únicamente hasta que un comando **SAVE RECORD** se ejecute para la tabla. Si cambia el registro actual (por ejemplo, por una búsqueda) antes de guardar el registro, el nuevo registro se pierde.

Nota: este comando no requiere tabla para estar en modo lectura/escritura. Se puede utilizar incluso cuando la tabla está en modo de sólo lectura (ver **Record Locking**).

Ejemplo

El siguiente ejemplo archiva los registros que tienen más de 30 días. Esta operación se realiza para la creación de registros en una tabla de archivo. Una vez termina la operación, los registros archivados son eliminados de la tabla [Cuentas]:

```
  ` Búsqueda de registros de más de 30 días
QUERY([Cuentas];[Cuentas]Introducido<(Current date 30))
For($vlRegistro;1;Records in selection([Cuentas]))  ` Bucle una vez por registro
   CREATE RECORD([Archivo])  ` Creación de un nuevo registro de archivo
   [Archive]Number:=[Cuentas]Numero  ` Copia de los campos en el archivo
   [Archive]Entered:=[Cuentas]Introducido
   [Archive]Amount:=[Cuentas]Cantidad
   SAVE RECORD([Cuentas])  ` Guardar el registro del archivo
   NEXT RECORD([Cuentas])  ` Mover el registro de cuenta siguiente
End for
DELETE SELECTION([Cuentas])  ` Borrar los registros de cuenta
```

DELETE RECORD

DELETE RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual borrar el registro actual, o Tabla por defecto, si se omite

Descripción

DELETE RECORD borra el registro actual de tabla en el proceso. Si no hay registro actual para tabla en el proceso, **DELETE RECORD** no tiene efecto. En un formulario, puede crear un botón Borrar registro en lugar de utilizar este comando.

Notas:

- si el registro actual es descargado de la memoria antes de llamar a DELETE RECORD (por ejemplo en respuesta a un **UNLOAD RECORD**), la selección actual de tabla está vacía después de la eliminación.
- el comando **DELETE RECORD** no hace nada cuando la tabla está en modo **READ ONLY**, independientemente de si el registro a borrar está bloqueado o no.

La eliminación de registros es una operación permanente y no puede deshacerse.

Si se borra un registro, el número del registro se reutilizará cuando se creen nuevos registros. No utilice el número del registro como identificador del registro si su base permite la eliminación de registros.

Ejemplo

El siguiente ejemplo borra un registro de un empleado. El código pregunta al usuario el número del empleado a borrar, busca el registro correspondiente y lo borra:

```
vBuscar:=Request("Número del empleado a borrar:") ` Obtiene un número de identificación del empleado
If(OK=1)
   QUERY([Empleado];[Empleado]ID =vBuscar) ` Buscar el empleado
   DELETE RECORD([Empleado]) ` Borrar el empleado
End if
```

DISPLAY RECORD

DISPLAY RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual mostrar el registro actual o Tabla por defecto, si se omite

Descripción

*El comando **DISPLAY RECORD** muestra el registro actual de tabla, utilizando el formulario de entrada actual. El registro se muestra hasta que un evento provoque un rediseño de la ventana. Tal evento puede ser la ejecución de un comando **ADD RECORD**, el regreso al formulario de entrada, o a la barra de menús. **DISPLAY RECORD** no hace nada si no hay registro actual.*

***DISPLAY RECORD** se utiliza con frecuencia para mostrar mensajes de progreso personalizados. También puede utilizarse para generar un presentación de diapositivas.*

*Si existe un método de formulario, se generará un evento **On Load**.*

***Advertencia:** no llame **DISPLAY RECORD** desde un proceso de conexión Web, porque el comando será ejecutado en el equipo del servidor Web de 4D y no en el equipo del cliente del navegador Web.*

Ejemplo

El siguiente ejemplo muestra una serie de registros como una presentación de diapositivas:

```
ALL RECORDS([Demo])  ` Selección de todos los registros
FORM SET INPUT([Demo];"Mostrar")  ` Designación del formulario a utilizar
For($vlRecord;1;Records in selection([Demo]))  ` Bucle a través de todos los registros
   DISPLAY RECORD([Demo])  ` Mostrar un registro
   DELAY PROCESS(Current process;180)  ` 3 segundos de pausa
   NEXT RECORD([Demo])  ` Pasar al siguiente registro
End for
```

DUPLICATE RECORD

DUPLICATE RECORD {(tabla)}

Parámetro	*Tipo*		*Descripción*
tabla	*Tabla*	➡	*Tabla del registro actual a duplicar o Tabla por defecto, si se omite*

Descripción

DUPLICATE RECORD *crea un nuevo registro para tabla que es un duplicado del registro actual. El nuevo registro se convierte en el registro actual. Si no hay registro actual,* ***DUPLICATE RECORD*** *no hace nada. Debe utilizar* ***SAVE RECORD*** *para guardar el nuevo registro.*

DUPLICATE RECORD *puede ejecutarse durante la entrada de datos. Esto permite duplicar el registro mostrado en pantalla. Recuerde que primero debe llamar* ***SAVE RECORD*** *para guardar los cambios realizados al registro original.*

Nota de compatibilidad*: a partir de la versión 11 de 4D, este comando no soporta subtablas.*

GOTO RECORD

GOTO RECORD ({tabla ;} posicion)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del registro de destino o Tabla por defecto, si se omite
posicion	Entero largo	➡	Número devuelto por número registro

Descripción

GOTO RECORD selecciona el registro actual de tabla. El parámetro registro es el número devuelto por la función **Record Number**. Después de ejecutar este comando, el registro es el único registro en la selección.

Si registro es inferior al número más pequeño en la base o superior al número más grande de la base, 4D genera un mensaje de error que indica que el número está fuera del intervalo. Si registro es igual al número de registro de un registro borrado, 4D devuelve el error -10503 y la selección queda vacía.

Ejemplo

Ver el ejemplo para **Record Number.**

Is new record

Is new record {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del registro a examinar o Tabla por defecto si se omite este parámetro
Resultado	Booleano	⮌	True si el registro está siendo creado, False si no

Descripción

*El comando **Is new record** devuelve **True** cuando el registro actual de tabla está siendo creado y aún no ha sido guardado en el proceso actual.*

***Nota de compatibilidad:** es posible obtener la misma información utilizando el comando existente **Record Number**, y probando si devuelve -3. Sin embargo, recomendamos utilizar **Is new record** en lugar de **Record Number** en este caso. De hecho, el comando **Is new record** asegura una mejor compatibilidad con las futuras versiones de 4D.*

***4D Server:** este comando devuelve un resultado diferente en el contexto del evento de formulario **On Validate** dependiendo de si se ejecuta en 4D (monopuesto) o 4D Client. En versión monopuesto, el comando devuelve **False** (el registro se considera como creado). En versión cliente/servidor, el comando devuelve **True** porque en este caso, el registro ya ha sido creado en el servidor pero la información no ha sido enviada aún al cliente.*

Ejemplo

Las dos siguientes instrucciones siguientes son idénticas. La segunda se recomienda para que el código sea compatible con las próximas versiones de 4D:

```
If(Record number([Tabla])=-3)  `No se recomienda
   ` ...
End if

If(Is new record([Tabla]))  `Recomendada
   ` ...
End if
```

Is record loaded

Is record loaded {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del registro a examinar o Tabla por defecto si se omite este parámetro
Resultado	Booleano	⮌	True si se carga el registro Si no False

Descripción

*El comando **Is record loaded** devuelve True si si el registro actual de tabla se carga en el proceso actual.*

***4D Server**: en principio, cuando las tablas están relacionadas por relaciones automáticas, los registros actuales de las tablas relacionadas se cargan automáticamente (ver **Relaciones**). Sin embargo, por razones de optimización, 4D Server sólo carga estos registros cuando es necesario, por ejemplo al leer o asignar un campo del registro relacionado. Como resultado, en este contexto, el comando **Is record loaded** devolverá False en modo remoto (devuelve True en modo local).*

Ejemplo

En lugar de utilizar las acciones automáticas "Siguiente registro" o "Registro anterior", puede escribir los métodos de objeto para estos botones para mejorar su operación. El botón "Siguiente" muestra el comienzo de la selección si el usuario está al principio de la selección y el botón "Anterior" muestra el final de la selección cuando el usuario está al comienzo de la selección.

```
  ` Método de objeto del botón "Anterior" (sin acción automática)
If(Form event=On Clicked)
   PREVIOUS RECORD([Grupo])
   If(Not(Is record loaded([Grupo])))
      GOTO SELECTED RECORD([Grupo];Records in selection([Grupo]))
  `Ir al último registro de la selección
   End if
End if

  ` Método de objeto del botón "Siguiente" (sin acción automática)
If(Form event=On Clicked)
   NEXT RECORD([Grupo])
   If(Not(Is record loaded([Grupo])))
      GOTO SELECTED RECORD([Grupos];1)
  `Ir al primer registro de la selección
   End if
End if
```

Modified record

Modified record {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	→	Tabla a probar si el registro actual se ha modificado o Tabla por defecto, si se omite
Resultado	Booleano	↩	El registro ha sido modificado (True), o El registro no ha sido modificado (False)

Descripción

Modified record *devuelve True si el registro actual de tabla se modificó pero no guardó; de lo contrario devuelve False. Esta función permite determinar rápidamente si el registro necesita ser guardado. En los formularios de entrada, puede efectuar la prueba antes de pasar al siguiente registro. Esta función siempre devuelve* ***TRUE*** *para un nuevo registro.*

Note que esta función siempre devuelve True en los siguientes contextos:

- *el registro actual es un nuevo registro,*
- *después de la ejecución de los comandos* ***PUSH RECORD*** *y* ***POP RECORD****,*
- *tan pronto como un valor ha sido asignado a un campo del registro, incluso si es el mismo valor que el anterior. Por ejemplo,* ***Modified record*** *devuelve True después de que se ejecute la siguiente instrucción:*

```
[Table_1]Field_1:=[Table_1]Field_1
```

Ejemplo

El siguiente ejemplo muestra una utilización típica de ***Modified record****:*

```
If(Modified record([Clientes]))
   SAVE RECORD([Clientes])
End if
```

POP RECORD

POP RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual desapilar el registro actual o Tabla por defecto, si se omite

Descripción

POP RECORD *carga el primer registro de la pila de registros de la tabla y lo vuelve el registro actual.*

Si apila un registro, luego crea una nueva selección actual que no incluye el registro apilado, y finalmente desapila el registro, entonces el registro actual no se encuentra en la selección actual. Para que el registro apilado esté en la selección actual, utilice ***ONE RECORD SELECT****.*

Si utiliza un comando que mueva el puntero del registro actual antes de guardar el registro, perderá la copia apilada en memoria.

Ejemplo

El siguiente ejemplo recupera el registro de un cliente en la pila:

```
POP RECORD([Clientes]) ` Desapilar el registro
```

PUSH RECORD

PUSH RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual apilar el registro actual o Tabla por defecto, si se omite

Descripción

PUSH RECORD *apila una copia del registro actual de tabla en la pila de registros de la tabla.* ***PUSH RECORD*** *puede ejecutarse antes de que se guarde un registro.*

Si apila un registro que fue desbloqueado, este registro permanece bloqueado para todos los otros procesos y usuarios hasta que lo desapile y descargue.

Nota de compatibilidad*: a partir de la versión 11 de 4D, este comando no soporta subtablas.*

Ejemplo

El siguiente ejemplo apila el registro de un cliente:

```
PUSH RECORD([Cliente]) ` Poner el registro del cliente en la pila
```

Record number

Record number {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	→	Tabla de la cual devolver el número del registro actual o Tabla por defecto, si se omite
Resultado	Entero largo	←	Número del registro actual

Descripción

Record number *devuelve el número del registro actual de tabla. Si no hay registro actual, como cuando el puntero del registro está antes o después de la selección actual,* ***Record number*** *devuelve –1. Si el registro es un nuevo registro que no ha sido guardado,* ***Record number*** *devuelve –3.*

Los números de registro pueden variar. Los números de registros borrados son reutilizados.
4D Server: *este comando devuelve un resultado diferente en el contexto del evento de formulario On Validate dependiendo de si se ejecuta en 4D en modo local o 4D en modo remoto. En versión monopuesto, el comando devuelve un número de registro (el registro se considera como creado). En versión cliente/servidor, el comando devuelve -3 porque en ese caso, el registros ya ha sido creado en el servidor pero la información no ha sido enviada al cliente.*

Nota: *se recomienda utilizar el comando* ***Is new record*** *para verificar si el registro está en proceso de creación.*

Ejemplo

El siguiente ejemplo guarda el número del registro actual y luego busca en la tabla si otro registro tiene el mismo valor:

```
$NumReg:=Record number([Personas])  ` Obtener el número del registro
QUERY([Personas];[Personas]Apellido=[Personas]Apellido)  ` ¿Alguien más con el mismo apellido?
  ` Mostrar una alerta con el nombre de las personas que tienen el mismo apellido
ALERT("Hay "+String(Records in selection([Personas])+" con ese apellido.")
GOTO RECORD([Personas];$NumReg)  ` Regresar al registro original
```

Records in table

Records in table {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	→	Tabla de la cual devolver el número total de registros o Tabla por defecto, si se omite
Resultado	Entero largo	↩	Número total de registros en la tabla

Descripción

Records in table *devuelve el número total de los registros que contiene tabla.* ***Records in selection*** *devuelve únicamente el número de registros en la selección actual. Si* ***Records in table*** *se utiliza dentro de una transacción, los registros creados durante la transacción serán tenidos en cuenta.*

Ejemplo

El siguiente ejemplo muestra una alerta que indica el número de registros en la tabla:

```
ALERT("Hay "+String(Records in table([Personas]))+" registros en la tabla.")
```

SAVE RECORD

SAVE RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del registro a guardar o Tabla por defecto, si se omite

Descripción

SAVE RECORD guarda el registro actual de tabla en el proceso actual. Si no hay registro actual, se ignora el comando SAVE RECORD.

Puede utilizar SAVE RECORD para guardar un registro creado o modificado por programación. Cuando un registro ha sido modificado y validado por el usuario en un formulario, no es necesario guardar con SAVE RECORD. Un registro que ha sido modificado por el usuario en un formulario, pero ha sido cancelado, aún pueden ser guardado con SAVE RECORD.

Estos son algunos casos donde es necesario SAVE RECORD:

- *Para guardar un nuevo registro creado con **CREATE RECORD** o **DUPLICATE RECORD***
- *Para guardar datos desde **RECEIVE RECORD***
- *Para guardar un registro modificado por un método*
- *Para guardar un registro que contiene un subregistro creado o modificado por uno de estos comandos **_o_ADD SUBRECORD**, **_o_CREATE SUBRECORD**, o **_o_MODIFY SUBRECORD***
- *Durante la entrada de datos, para guardar el registro mostrado antes de llamar un comando que cambia el registro actual*
- *Durante la entrada de datos, para guardar el registro actual*

No debe ejecutar SAVE RECORD en el evento de formulario On Validate de un registro que ha sido aceptado. Si lo hace, el registro se guardará dos veces.

***Nota:** guardar un registro que contiene campos objeto editados generalmente requiere que notifique explícitamente a 4D antes de llamar a **SAVE RECORD**. Para más información, consulte la sección **Guardar campos objeto**.*

Ejemplo

El siguiente ejemplo es parte de un método que lee registros de un documento. En esta parte del código, se recibe un registro, y luego, si se recibe correctamente, se guarda el registro:

```
RECEIVE RECORD([Clientes])  ` Recepción del registro a partir del disco
If(OK=1)  ` Si el registro se recibe correctamente···
   SAVE RECORD([Clientes])  ` guardar
End if
```

Sequence number

Sequence number {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	→	Tabla a numerar automáticamente o Tabla por defecto, si se omite
Resultado	Entero largo	←	Número automático

Descripción

Sequence number *devuelve el próximo número automático de tabla. El número de secuencia es único para cada tabla. Este número es único para cada tabla. Este valor no se repite y se incrementa por cada nuevo registro creado en la tabla (*).*

(*) Por razones de optimización, la numeración automática sólo se inicia en la primera llamada del comando ***Sequence number*** *o de una funcionalidad que tenga acceso al número de secuencia (ver abajo). Además, la numeración se puede modificar utilizando el comando* ***SET DATABASE PARAMETER****. Por lo tanto, el valor devuelto no se debe considerar como el número de registros creados en tabla.*

Por defecto, la numeración comienza en 1. Puede cambiar la numeración para una tabla utilizando el comando ***SET DATABASE PARAMETER****.*

Nota: *si no hay un registro actual y la numeración se ha modificado a través del comando* ***SET DATABASE PARAMETER****, este número se reserva para la próxima creación de registro, pero sólo será devuelto por la función*

Sequence number *cuando el comando* ***SAVE RECORD*** *se haya llamado.*
La función ***Sequence number*** *es útil en los siguientes casos:*

- *Si el número de secuencia es mayor que 1*
- *Si el número de secuencia es parte de un código, por ejemplo un código de número de parte.*

Para almacenar el número de secuencia con la ayuda de un método, cree un campo de tipo Entero largo en la tabla y asigne el número de secuencia al campo.

El número de secuencia devuelto por esta función para la tabla es el mismo número que el generado al seleccionar la opción ***Autoincrementar*** *para un campo de la tabla utilizando el inspector de estructura o al asignado utilizando el símbolo #N como valor por defecto para un campo de la tabla en un formulario. Para mayor información sobre la asignación de valores, consulte el Manual de Diseño de 4D.*

Nota: *la numeración automática también puede asignarse vía el atributo SQL AUTO_INCREMENT.*

Si la numeración debe comenzar en un valor diferente de 1, simplemente añada la diferencia a ***Sequence number****. Por ejemplo, si el número de secuencia debe comenzar en 1 000, puede utilizar la siguiente línea de código para asignar el número:*

```
[Table1]Seq Field :=Sequence number([Table1])+999
```

Ejemplo

El siguiente ejemplo es parte de un método de formulario. Estas líneas de código prueban si se trata de un nuevo registro (si el número de factura es igual a una cadena vacía). Si es un nuevo registro, el método asigna un número de factura. El número de factura está formado por dos partes: el número de secuencia, y el identificador del operador, el cual fue introducido al abrir la base. El número de secuencia es formateado como una cadena de cinco caracteres:

```
  ` Si es una nueva factura, crear un número de factura
If([Facturas]NumFactura="")
  ` El número de factura es una cadena que se termina por el número de referencia del usuario.
   [Facturas]NumFactura:=String(Sequence number;"00000")+[Facturas]Usuario
End if
```

Registros (bloqueo)

- *Record Locking*
- *Get locked records info*
- *LOAD RECORD*
- *Locked*
- *LOCKED BY*
- *READ ONLY*
- *Read only state*
- *READ WRITE*
- *UNLOAD RECORD*

Record Locking

4D y 4D Server administran automáticamente las bases evitando conflictos entre procesos o entre usuarios. Dos usuarios o dos procesos no pueden modificar al mismo tiempo el mismo registro u el mismo objeto. El segundo usuario o proceso puede acceder simultáneamente al registro u objeto en modo sólo lectura.

Hay muchas razones para utilizar los comandos multiusuario:

- *Modificación de registros por programación.*
- *Utilización de una interfaz de usuario personalizada para operaciones multiusuario.*
- *Almacenamiento de modificaciones relacionadas con una transacción.*

Hay tres conceptos importantes a tener en cuenta cuando se utilizan comandos en una base multiproceso:

- *En un proceso, cada tabla está en modo sólo lectura o lectura/escritura.*
- *Los registros se bloquean cuando son cargados y se desbloquean cuando son descargados.*
- *Un registro bloqueado no puede ser modificado.*

*En las siguientes secciones como convención, la persona que efectúa una operación en la base multiusuarios es el **usuario local**. Las otras personas que utilizan la base son los **otros usuarios**. La discusión es desde el punto de vista del usuario local. De la misma forma, desde el punto de vista multiproceso, el proceso que se ejecuta una operación en la base es el **proceso actual**. Todo otro proceso en curso de ejecución está diseñado como **otro proceso**. La discusión es desde el punto de vista del proceso actual.*

Bloqueo de registros

Un registro bloqueado no puede ser modificado por el usuario local o el proceso actual. Un registro bloqueado puede ser cargado, pero no modificado. Un registro se bloquea cuando uno de los otros usuarios o procesos carga el registro para efectuar una modificación o cuando el registro está apilado. Sólo el usuario que modifica el registro ve el registro como desbloqueado. Todos los otros usuarios y procesos ven el registro como bloqueado y por lo tanto no disponible para modificación. Una tabla debe estar en modo lectura/escritura para que un registro se cargue como desbloqueado.

Modo sólo lectura y lectura/escritura

*Cada tabla de una base está en modo lectura/escritura o en modo sólo lectura para cada usuario y proceso de la base. **Sólo lectura** significa que los registros de la tabla pueden ser cargados pero no modificados. **Lectura/escritura** significa que los registros de la tabla pueden ser cargados y modificados si ningún otro usuario/proceso ha bloqueado el registro previamente.*

Note que si cambia el estado de una tabla, el cambio toma efecto para el siguiente registro cargado. Si ya hay un registro cargado cuando cambia el estado de la tabla, el registro no se afecta por el cambio de estado.

Estado sólo lectura

Cuando una tabla está en modo sólo lectura y se carga un registro, el registro está siempre bloqueado. En otras palabras, el registro puede mostrarse, imprimirse y utilizarse, pero no modificarse.

*Note que el modo sólo lectura aplica únicamente a la edición de registros existentes. El estado sólo lectura no afecta la creación de nuevos registros. Puede añadir registros a una tabla sólo lectura utilizando los comandos **CREATE RECORD** y **ADD RECORD** o los comandos de menús del entorno Diseño (en este caso, los registros, en creación son bloqueados para todos los otros usuarios/procesos). Note que el comando **ARRAY TO SELECTION** no se ve afectado por el estado sólo lectura ya que permite crear y modificar registros.*

*4D define automáticamente una tabla en modo sólo lectura para los comandos que no requieren acceder en escritura a los registros. Estos comandos son: **DISPLAY SELECTION, DISTINCT VALUES, EXPORT DIF, EXPORT SYLK, EXPORT TEXT, _o_GRAPH TABLE, PRINT SELECTION, PRINT LABEL, QR REPORT, SELECTION TO ARRAY, SELECTION RANGE TO ARRAY**.*

*Puede saber en cualquier momento el estado de una tabla utilizando la función **Read only state**.*

Antes de ejecutar cualquiera de estos comandos, 4D guarda el estado actual de la tabla (sólo lectura o lectura/escritura) para el proceso actual. Después de ejecutar el comando, el estado inicial se restablece.

Estado lectura/escritura

Cuando una tabla está en lectura/escritura y se carga un registro, el registro estará desbloqueado si ningún otro usuario ha bloqueado el registro primero. Si el registro está bloqueado por otro usuario, el registro se carga como un registro bloqueado que no puede ser modificado por el usuario local.

Una tabla debe estar en modo lectura/escritura y el registro cargado para que sea desbloqueada y por lo tanto modificable.

*Si un usuario carga un registro de una tabla en modo lectura/escritura, ningún otro usuario puede cargar ese registro para modificación. Sin embargo, otros usuarios pueden añadir registros a la tabla, bien sea a través de los comandos **CREATE RECORD** o **ADD RECORD** o manualmente en el entorno Diseño.*

El modo lectura/escritura es el estado por defecto para todas las tablas cuando una base se abre y se inicia un nuevo proceso.

Cambiar el estado de una tabla

*Puede utilizar los comandos **READ ONLY** y **READ WRITE** para cambiar el estado de una tabla. Si quiere cambiar el estado de una tabla para volver un registro de sólo lectura o lectura/escritura, puede ejecutar el comando antes de cargar el registro. Todo registro ya cargado no se ve afectado por los comandos **READ ONLY** y **READ WRITE**.*

Cada proceso tiene su propio estado (sólo lectura o lectura/escritura) para cada tabla en la base.
*Por defecto, si no se utiliza el comando **READ ONLY**, todas las tablas están en modo lectura/escritura.*

Cargar, modificar y descargar registros

Para que el usuario local pueda modificar un registro, la tabla debe estar en modo lectura/escritura y el registro debe cargarse y desbloquearse.

*Cada uno de los comandos que carga un registro actual (si hay uno), tales como **NEXT RECORD**, **QUERY**, **ORDER BY**, **RELATE ONE**, etc., define el estado bloqueado o desbloqueado del registro. El registro se carga en función del estado actual de su tabla (sólo lectura o lectura/escritura) y de su disponibilidad. Un registro también puede cargarse de una tabla relacionada por uno de los comandos que provoca una relación automática.*

Si una tabla está en modo lectura únicamente, para un proceso o usuario, todo registro de esta tabla se carga en modo lectura únicamente, lo que significa que no podrá ser modificado o eliminado por el proceso o usuario. Este modo se recomienda para la visualización o recuperación de datos, ya que no impide que otros usuarios o procesos accedan a los registros de esta tabla en modo lectura/escritura si es necesario.

Si una tabla está en modo lectura/escritura, todo registro cargado de esta tabla está desbloqueado sólo si ningún otro usuario ha bloqueado el registro primero. Un registro desbloqueado puede ser modificado y guardado. Una tabla debe ser colocada en modo lectura/escritura antes de que un registro necesite ser cargado, modificado, y luego guardado.
*Si el registro debe modificarse, utilice la función **Locked** para probar si el registro está bloqueado por otro usuario. Si un registro está bloqueado (**Locked** devuelve True), cargue el registro con el comando [#cmd id="52"/] y pruebe nuevamente si el registro está bloqueado o no. Esta secuencia debe continuar hasta que el registro sea desbloqueado (**Locked** devuelve False).*

*Cuando termine las modificaciones a un registro, el registro debe liberarse (y por lo tanto desbloquearse para los otros usuarios) con **UNLOAD RECORD**. Si no se descarga un registro, permanecerá bloqueado para todos los demás usuarios hasta que se seleccione un registro actual diferente. Cambiar el registro actual de una tabla desbloquea automáticamente el registro actual anterior. Debe llamar explícitamente **UNLOAD RECORD** si no cambia el registro actual. Este principio aplica a los registros existentes. Cuando se crea un nuevo registro, puede guardarse sin importar el estado de la tabla a la cual pertenece.*

***Nota:** cuando se utiliza en una transacción, el comando **UNLOAD RECORD** descarga el registro actual sólo para el proceso que administra la transacción. Para los otros procesos, el registro permanece bloqueado hasta que la transacción no sea validada (o cancelada).*

*Utilice el comando **LOCKED BY** para ver que usuario y/o proceso tiene un registro bloqueado.*

***Nota:** una buena práctica es colocar todas las tablas en modo de sólo lectura cuando se inicia cada proceso (utilizando la sintaxis **READ ONLY(*)**) a continuación, se pone cada tabla en modo lectura/escritura sólo cuando sea necesario. El acceso a las tablas en modo de sólo lectura es más rápido y más económico en memoria. Por otra parte, cambiando el estado de una tabla se optimiza en modo cliente/servidor, ya que no causa ningún tráfico de red adicional: la información sólo se envía al servidor cuando se ejecuta un comando que requiere un acceso adecuado a la tabla.*

Bucles para cargar registros no bloqueados

El siguiente ejemplo muestra el bucle más simple para cargar un registro no bloqueado:

```
READ WRITE([Customers]) ` Define el estado de la tabla en lectura/escritura
Repeat ` Hace un bucle hasta que el registro esté desbloqueado
   LOAD RECORD([Customers]) ` Carga el registro y lo bloquea
Until(Not(Locked([Customers])))
 ` Hacer algo para el registro acá
READ ONLY([Customers]) ` Define el estado de la tabla en modo sólo lectura
```

El bucle continúa hasta que el registro sea desbloqueado.

Un bucle como este se utiliza únicamente cuando es poco probable que el registro esté bloqueado por otra persona, ya que el usuario tendría que esperar hasta que el bucle termine. Este bucle no se utiliza a menos que el registro sólo sea modificable por un método.

El siguiente ejemplo utiliza el bucle anterior para cargar un registro desbloqueado y modificar el registro:

```
READ WRITE([Inventario])
Repeat ` Bucle hasta que el registro sea desbloqueado
   LOAD RECORD([Inventario]) ` Cargar el registro y lo define como bloqueado
Until(Not(Locked([Inventario])))
[Inventory]Part Qty:=[Inventario]Part Qty-1 ` Modifica el registro
SAVE RECORD([Inventario]) ` Guarda el registro
UNLOAD RECORD([Inventario]) ` Permite que otros usuarios lo modifiquen
READ ONLY([Inventario])
```

*El comando **MODIFY RECORD** notifica automáticamente al usuario si un registro está bloqueado y evita que el registro sea modificado. El siguiente ejemplo evita esta notificación automática probando el registro primero con la función **Locked**. Si el registro está bloqueado, el usuario puede cancelar.*

Este ejemplo prueba eficientemente si el registro actual está bloqueado para la tabla [Comandos]. Si está bloqueado, el proceso es retrasado por el método por un segundo. Esta técnica puede utilizarse en un desarrollo multiusuario o multiproceso:

```
Repeat
   READ ONLY([Comandos]) ` No necesita lectura/ escritura por el momento
```

```
   QUERY([Comandos])
` Si la búsqueda se terminó y lo registros son devueltos
   If((OK=1) & (Records in selection([Comandos])>0))
      READ WRITE([Comandos]) ` Coloque la tabla en modo lectura/escritura
      LOAD RECORD([Comandos])
      While(Locked([Comandos]) & (OK=1)) `Si el registro está bloqueado,
` bucle hasta que el registro sea liberado
` ¿Quién bloqueó el registro?
         LOCKED BY([Comandos];$Procesos;$Usuario;$UsuarioSesion;$Nombre)
         If($Proceso=-1) ` ¿El registro ha sido borrado?
            ALERT("El registro ha sido borrado en el intervalo.")
            OK:=0
         Else
            If($Usuario="") ` ¿Está en modo monusuario?
               $Usuario:="usted"
            End if
            CONFIRM("El registro está siendo utilizado por "+$Usuario+" en el proceso "+$Nombre+" Proceso.")
            If(OK=1) ` Si quiere esperar algunos segundos
               DELAY PROCESS(Current process;120) ` Espere algunos segundos
               LOAD RECORD([Comandos]) ` Trate de cargar el registro
            End if
         End if
      End while
      If(OK=1) ` El registro está desbloqueado
         MODIFY RECORD([Comandos]) ` Puede modificar el registro
         UNLOAD RECORD([Comandos])
      End if
      READ ONLY([Comandos]) ` Volver a modo sólo lectura
      OK:=1
   End if
Until(OK=0)
```

Uso de comandos en entornos Multi-usuario o Multi-proceso

Ciertos comandos del lenguaje realizan acciones particulares cuando encuentran un registro bloqueado.
Esta es la lista de estos comandos y sus acciones cuando encuentran un registro bloqueado.

- ***MODIFY RECORD**: muestra una caja de diálogo indicando que el registro está en uso. El registro no se muestra, por lo tanto el usuario no puede modificarlo. En el entorno Diseño, el registro se muestra en estado sólo lectura.*
- ***MODIFY SELECTION**: se comporta normalmente excepto cuando el usuario hace doble clic en un registro para modificarlo. **MODIFY SELECTION** muestra una caja de diálogo indicando que el registro está en uso y luego permite el acceso al registro en modo sólo lectura.*
- ***APPLY TO SELECTION**: carga un registro bloqueado, pero no lo modifica. **APPLY TO SELECTION** puede utilizarse para leer información de una tabla sin tener cuidados especiales. Si el comando encuentra un registro bloqueado, el registro se coloca en el conjunto sistema **LockedSet**.*
- ***DELETE SELECTION**: no borra los registros bloqueados; simplemente los ignora. Si el comando encuentra un registro bloqueado, el registro se pone en el conjunto sistema **LockedSet**.*
- ***DELETE RECORD**: este comando se ignora si el registro está bloqueado. No se devuelve ningún error. Debe probar que el registro está desbloqueado antes de ejecutar este comando.*
- ***SAVE RECORD**: este comando se ignora si el registro está bloqueado. No se devuelve ningún error. Debe probar que el registro está desbloqueado antes de ejecutar este comando.*
- ***ARRAY TO SELECTION**: no guarda los registros bloqueados. Si el comando encuentra un registro bloqueado, el registro se pone en el conjunto sistema **LockedSet**.*
- ***GOTO RECORD**: en una base multiusuarios/multiprocesos los registros pueden ser añadidos o borrados por otros usuarios, por lo tanto los números de los registros pueden variar. Sea prudente cuando referencie un registro directamente un registro por número en una base multiusuarios.*
- ***Conjuntos**: sea especialmente cuidadoso con los conjuntos, ya que la información en la que se basa el conjunto puede ser cambiada por otro usuario o proceso.*

Get locked records info

Get locked records info (laTabla) -> Resultado

Parámetro	Tipo		Descripción
laTabla	Tabla	➡	Tabla de la cual obtener los registros bloqueados
Resultado	Objeto	⮌	Descripción de los registros bloqueados (si los hay)

Descripción

*El comando **Get locked records info** devuelve un objeto que contiene diferente información sobre los registros bloqueados actualmente en laTabla.*

El objeto devuelto contiene una propiedad "registros", que es un array de objetos:

```
{
    "records": [
        objeto descripción,
        (…)
    ]
}
```

Cada elemento del array "description object" identifica un registro bloqueado en la tabla especificada y contiene las siguientes propiedades:

Propiedad	Tipo	Descripción
contextID	UUID (Cadena)	UUID del contexto de la base responsable del bloqueo
contextAttributes	Objeto	Objeto que contiene la misma información que el comando **LOCKED BY** aplicado al registro, la diferencia es que **Get locked records info** devuelve el nombre del usuario definido en el sistema y no el del usuario 4D, así como también información adicional (ver más adelante).
recordNumber	Entero largo	Número de registro del registro bloqueado

*Nota: por consistencia, el comando LOCKED ATTRIBUTES ha sido renombrado **LOCKED BY** a partir de 4D v14 R3.*

El objeto contextAttributes se compone de las siguientes propiedades:

Propiedad	Tipo	Descripción
task_id	Número	Número de referencia del proceso
user_name	Cadena	Nombre del usuario definido en el sistema operativo
user4d_id	Número	Número del usuario 4D(*)
host_name	Cadena	Nombre de la máquina local
task_name	Cadena	Nombre del proceso
client_version	Número	Versión de la aplicación cliente
Únicamente cuando el comando se ejecuta en 4D Server y si el bloqueo del registro proviene de un 4D remoto:		
is_remote_context	Boolean	Indica si un el origen del bloqueo es un 4D remoto (siempre true ya que de lo contrario no está presente)
client_uid	UUID (Cadena)	UUID del 4D remoto en el origen del bloqueo

(*) Puede obtener el nombre del usuario 4D a partir del valor de user4d_id utilizando el siguiente código:

```
GET USER LIST($arrNames;$arrIDs)
$User4DName:=Find in array($arrIDs;user4d_id)
```

***Nota**: el comando funciona únicamente con 4D y 4D Server. Siempre devuelve un objeto no válido cuando se llama desde un 4D remoto o un componente. Sin embargo, se le puede llamar en estos contextos, si se activa la opción "Ejecutar en el servidor". En este caso, el objeto devuelto contendrá, respectivamente, la información sobre el servidor o la base local.*

Ejemplo

Ejecute el siguiente código:

```
$vOlocked :=Get locked records info([Table])
```

Si dos registros están bloqueados en la tabla [Table], el siguiente objeto se devuelve en $vOlocked:

```
{
    "records": [
        {
            "contextID": "A9BB84C0E57349E089FA44E04C0F2F25",
             "contextAttributes": {
```

```
                "task_id": 8,
                "user_name": "roland",
                "user4d_id": 1,
                "host_name": "iMac de roland",
                "task_name": "P_RandomLock",
                "client_version": -1342106592
            },
            "recordNumber": 1
        },
        {
            "contextID": "8916338D1B8A4D86B857D92F593CCAC3",
            "contextAttributes": {
                "task_id": 9,
                "user_name": "roland",
                "user4d_id": 1,
                "host_name": "iMac de roland",
                "task_name": "P_RandomLock",
                "client_version": -1342106592
            },
            "recordNumber": 2
        }
    ]
}
```


Si el código se ejecuta en un 4D Server y el bloqueo es causado por una máquina cliente remota, el siguiente objeto es devuelto en $vOlocked:


```
{
    "records": [
        {
            "contextID": "B0EC087DC2FA704496C0EA15DC011D1C",
            "contextAttributes": {
                "task_id": 2,
                "user_name": "achim",
                "user4d_id": 1,
                "host_name": "achim-pcwin",
                "task_name": "P_RandomLock",
                "is_remote_context": true,
                "client_uid": "0696E66F6CD731468E6XXX581A87554A",
                "client_version": -268364752
            },
            "recordNumber": 1
        }
    ]
}
```

LOAD RECORD

LOAD RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla de la cual cargar el registro actual o Tabla por defecto si se omite

Descripción

LOAD RECORD *carga el registro actual de tabla. Si no hay registro actual,* ***LOAD RECORD*** *no tienen ningún efecto.*
Puede utilizar la función ***Locked*** *para determinar si puede modificar el registro:*

- *Si la tabla está en modo sólo lectura, la función* ***Locked*** *devuelve TRUE, y no es posible modificar el registro.*
- *Si la tabla está en modo lectura/escritura pero el registro ha sido bloqueado, el registro será sólo lectura, y no es posible modificar el registro.*
- *Si la tabla está en modo lectura/escritura y el registro no está bloqueado, es posible modificar el registro en el proceso actual. La función* ***Locked*** *devuelve TRUE para todos los otros usuarios y procesos.*

Nota: *si el comando* ***LOAD RECORD*** *se ejecuta después de un* ***READ ONLY****, el registro se libera automáticamente y se carga sin tener que utilizar el comando* ***UNLOAD RECORD****.*

Generalmente, no es necesario utilizar el comando ***LOAD RECORD****, porque los comandos como* ***QUERY****,* ***NEXT RECORD****,* ***PREVIOUS RECORD****, etc., cargan automáticamente el registro actual.*

En entornos multiusuario y multiprocesos, cuando necesite modificar un registro existente, debe acceder a la tabla (a la cual pertenece el registro) en modo lectura/escritura. Si un registro está bloqueado y no puede ser cargado, ***LOAD RECORD*** *le permite intentar cargar el registro nuevamente más tarde. Utilizando* ***LOAD RECORD*** *en un bucle, puede esperar hasta que el registro esté disponible en modo lectura/escritura.*

Consejo: *el comando* ***LOAD RECORD*** *puede ser utilizado para recargar el registro actual en el contexto de un formulario de entrada. Todos los datos modificados son reemplazados por los valores anteriores. En ese caso, el comando* ***LOAD RECORD*** *efectúa en cierto modo una cancelación general de la entrada.*

Locked

Locked {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	→	Tabla en la cual verificar si el registro actual está bloqueado o Tabla por defecto, si se omite
Resultado	Booleano	↩	El registro está bloqueado (TRUE), o El registro no está bloqueado (FALSE)

Descripción

Locked prueba si el registro actual de tabla está bloqueado. Utilice esta función para saber si un registro está bloqueado o no; luego tome la acción más conveniente, tal como darle al usuario la opción de esperar que el registro sea liberado o de anular al operación.

Si **Locked** devuelve TRUE, el registro no puede guardarse porque está bloqueado por otro usuario o proceso, o está apilado en el proceso actual. En este caso, utilice **LOAD RECORD** para recargar el registro hasta que **Locked** devuelva FALSE.

Si**Locked** devuelve FALSE, el registro está desbloqueado, lo que significa que está bloqueado para los otros usuarios. Sólo el usuario local o el proceso actual puede modificar y guardar el registro. Una tabla debe estar en modo lectura/escritura si quiere modificar los registros que contiene.

Si trata de cargar un registro que ha sido borrado, **Locked** devuelve TRUE. Para evitar esperar por un registro que no existe, utilice el comando **LOCKED BY**. Si el registro ha sido borrado, el comando **LOCKED BY** devuelve -1 en el parámetro proceso.

Nota: Locked devuelve False cuando no hay un registro actual en tabla, en otras palabras, cuando **Record number** devuelve -1.

Durante una transacción, **LOAD RECORD** y **Locked** se utilizan con frecuencia para probar la disponibilidad de los registros. Si un registro está bloqueado, es común cancelar la transacción.

LOCKED BY

LOCKED BY ({tabla ;} proceso ; usuario4D ; sesionUsuario ; nombreProceso)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a verificar si el registro está bloqueado o Tabla por defecto, si se omite
proceso	Entero largo	⬅	Número de referencia del proceso
usuario4D	Cadena	⬅	Nombre del usuario 4D
sesionUsuario	Cadena	⬅	Nombre del usuario que inició la sesion de trabajo
nombreProceso	Cadena	⬅	Nombre del proceso

Descripción

LOCKED BY devuelve la información sobre el usuario y el proceso que tiene bloqueado un registro. El número del proceso (*), el nombre de usuario en la aplicación 4D y en el sistema así como el nombre del proceso son devueltos en las variables proceso, usuario4D, sesionUsuario, y nombreProceso. Puede utilizar esta información en una caja de diálogo personalizada para advertir al usuario cuando un registro está bloqueado.

(*) Este es el número del proceso en la máquina donde se ejecuta el código que origina el bloqueo del registro. En el caso de un trigger o un método que se ejecuta en el servidor, se devuelve el número del proceso "gemelo" en la máquina servidor. En el caso de un método que se ejecuta en una aplicación remota, se devuelve el número del proceso en la máquina remota.

Si el registro no está bloqueado, proceso devuelve 0 y usuario, equipo, y nombreProceso devuelven cadenas vacías. Si trata de cargar en modo lectura/escritura un registro que ha sido borrado, proceso devuelve -1 y usuario, equipo, y nombreProceso devuelven cadenas vacías.

En modo monousuario, este comando devuelve los valores en proceso y nombreProceso únicamente si un registro está bloqueado. Los valores devueltos en usuario y equipo son cadenas vacías.

En modo Cliente/Servidor, el número de proceso devuelto es el número del proceso en el servidor.

El parámetro Usuario devuelto corresponde al nombre del usuario definido en el editor de contraseñas de 4D. Si no hay sistema de contraseñas, devuelve "Diseñador".

El parámetro equipo devuelto corresponde al nombre del usuario que abrió la sesión en el equipo cliente (este nombre aparece en la ventana de administración de 4D Server para cada proceso abierto).

READ ONLY

READ ONLY {(tabla | *)}

Parámetro	*Tipo*		*Descripción*
*tabla | **	*Tabla, Operador*	➡	*Tabla a definir en modo sólo lectura o * para todas las tablas o Tabla por defecto, si se omite*

Descripción

***READ ONLY** cambia el estado de tabla a modo sólo lectura para el proceso en el cual se llama al comando. Todos los registros cargados posteriormente están bloqueados, y no se puede realizar ninguna modificación. Si se pasa el parámetro opcional *, todas las tablas se cambian a modo sólo lectura.*

*Utilice **READ ONLY** cuando no necesite modificar los registros.*

***Nota:** este comando no es retroactivo. Los privilegios de lectura/escritura para un registro son definidos por los privilegios de la tabla en el momento en que se carga el registro. Para cargar un registro en modo sólo lectura cuando la tabla está en modo lectura/escritura, primero debe cambiar el estado de la tabla a modo sólo lectura.*

Read only state

Read only state {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla para la cual probar el estado sólo lectura o Tabla por defecto, si se omite
Resultado	Booleano	⮌	El acceso a la tabla es sólo lectura (TRUE), o El acceso a la tabla es lectura/escritura (FALSE)

Descripción

Esta función se utiliza para probar si tabla está en modo sólo lectura en el proceso en el que se llamó a la función. ***Read only state*** *devuelve TRUE si el estado de tabla es sólo lectura y FALSE si el estado de tabla es lectura/ escritura.*

Ejemplo

El siguiente ejemplo prueba el estado de la tabla [Facturas]. Si el estado de la tabla [Facturas] es sólo lectura, se aplica el modo lectura/escritura y se carga nuevamente el registro actual.

```
If (Read only state ([Facturas]))
   READ WRITE ([Facturas])
   LOAD RECORD ([Facturas])
End if
```

Nota: *el registro actual se carga nuevamente para permitirle al usuario modificarlo. Un registro cargado anteriormente en modo sólo lectura permanecerá bloqueado hasta que se recargue en modo lectura/escritura.*

READ WRITE

READ WRITE {(tabla | *)}

Parámetro	Tipo		Descripción
tabla | *	Tabla, Operador	➡	Tabla a definir en modo lectura/escritura o * para todas las tablas o Tabla por defecto si se omite

Descripción

READ WRITE *cambia el estado de tabla a modo lectura/escritura para el proceso en el cual se llama al comando. Si pasa el parámetro opcional *, todas las tablas pasan a modo lectura/escritura.*

Después de llamar a ***READ WRITE****, cuando se carga un registro, el registro está desbloqueado si ningún otro usuario ha bloqueado el registro. Este comando no cambia el estado del registro cargado actualmente, sólo el de los registros cargados posteriormente.*

Por defecto, todas las tablas están en modo lectura/escritura.

Utilice ***READ WRITE*** *cuando tenga que modificar un registro y guardar los cambios. También puede utilizar* ***READ WRITE*** *cuando quiera bloquear un registro para los otros usuarios, incluso cuando no esté realizando cambios. Colocar una tabla en modo lectura/escritura evita que otros usuarios editen la tabla. Sin embargo, los otros usuarios pueden crear nuevos registros.*

Nota: *este comando no es retroactivo. Los privilegios de lectura/escritura para un registro son definidos por los privilegios de la tabla en el momento en que se carga el registro. Para cargar un registro en modo lectura/escritura de una tabla sólo lectura, primero debe cambia el estado de la tabla a lectura/escritura.*

UNLOAD RECORD

UNLOAD RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual descargar el registro o Tabla por defecto, si se omite

Descripción

UNLOAD RECORD descarga el registro actual de tabla.

Si el registro no está bloqueado para el usuario local (bloqueado para los otros usuarios), **UNLOAD RECORD** desbloquea el registro para los otros usuarios.

Aunque **UNLOAD RECORD** descarga el registro de la memoria, el registro permanece como registro actual. Cuando otro registro se convierte en el registro actual, el registro actual anterior se descarga automáticamente y se desbloquea para los otros usuarios. Siempre ejecute este comando cuando haya terminado de modificar un registro y quiera que esté disponible para otros usuarios, mientras permanece el registro como su registro actual.

Si un registro tiene una cantidad importante de datos, de campos de imagen, o de documentos externos (tales como documentos 4D Write o 4D Draw), es preferible no almacenar el registro actual en memoria a menos que necesite modificarlo. En ese caso, utilice el comando **UNLOAD RECORD,** de esta manera, puede conservar el registro actual sin que esté en memoria. De esta forma libera la memoria ocupada por el registro, pero no tiene acceso a los valores almacenados en los campos. Si más adelante necesita acceder a los valores del registro, utilice el comando **LOAD RECORD**.

Nota: cuando se utiliza en una transacción, el comando **UNLOAD RECORD** descarga el registro actual únicamente para el proceso que genera la transacción. Para otros procesos, el registro permanece bloqueado siempre que la transacción no hay sido validada (o cancelada).

Relaciones

- *Relaciones*
- *CREATE RELATED ONE*
- *GET AUTOMATIC RELATIONS*
- *GET FIELD RELATION*
- *OLD RELATED MANY*
- *OLD RELATED ONE*
- *RELATE MANY*
- *RELATE MANY SELECTION*
- *RELATE ONE*
- *RELATE ONE SELECTION*
- *SAVE RELATED ONE*
- *SET AUTOMATIC RELATIONS*
- *SET FIELD RELATION*

Relaciones

*Los comandos de este tema, en particular **RELATE ONE** y **RELATE MANY**, establecen y administran las relaciones entre las tablas, tanto las relaciones automáticas como las relaciones manuales. Antes de utilizar los comandos de este tema, por favor consulte el Manual de Diseño de 4D para información sobre la creación de relaciones entre tablas.*

Uso de relaciones automáticas entre tablas por programación

Dos tablas pueden estar relacionadas por una relación automática. En general, cuando se crea una relación automática, los registros relacionados son cargados y seleccionados en la tabla relacionada. Muchas operaciones hacen que la relación se establezca.

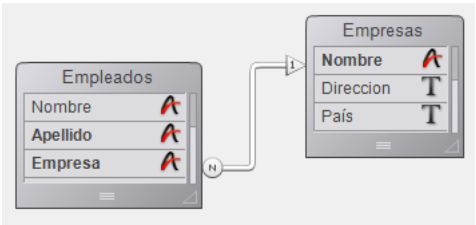
Estas operaciones incluyen:

- *Entrada de datos*
- *Lista de registros en la pantalla en un formulario de salida*
- *Informes*
- *Operaciones en una selección de registros como las búsquedas, las ordenaciones y las fórmulas.*

Para mejorar el rendimiento, cuando 4D activa las relaciones automáticas, sólo un registro se convierte en el registro actual para la tabla. Para cada una de las operaciones enumeradas anteriormente, el registro relacionado se carga de acuerdo a los siguientes principios:

- *Si una relación selecciona un solo registro de la tabla relacionada, el registro se carga del disco.*
- *Si una relación selecciona más de un registro de una tabla relacionada, una nueva selección de registros se crea para esa tabla, y la el primer registro en esa selección se carga del disco.*

Por ejemplo, en la estructura que se muestra a continuación, si un registro para la tabla [Empleados] se carga y muestra para la entrada de datos, el registro relacionado en la tabla [Empresas] se selecciona y se carga. Igualmente, si un registro para la tabla [Empresas] se carga y muestra para la entrada de datos, los registros relacionados con la tabla [Empleados] son seleccionados.



*En esta estructura, la tabla [Empleados] es la **Tabla muchos**, y la tabla [Empresas] es la **Tabla uno**. Para recordar este concepto, piense en que "hay muchos empleados relacionados con una empresa" y "cada empresa tiene muchos empleados."*

*De manera similar, el campo Empresa de la tabla [Empleados] es el **Campo muchos**, y el campo Nombre en la tabla [Empresas] es el **Campo uno**.*

No siempre es posible que el campo relacionado sea único. Por ejemplo, el campo [Empresas]Nombre puede tener varios registros de nombres de empresas iguales. En esta situación la no unicidad puede manejarse fácilmente creando una relación, la cual siempre será única, u otro campo en la tabla relacionada. Este campo puede ser un número de identificación de la empresa.

La siguiente tabla lista los comandos que utilizan las relaciones automáticas para cargar los registros relacionados durante su ejecución. Todos estos comandos utilizan relaciones automáticas de Muchos a Uno. Sólo los comandos señalados con un Sí activan las relaciones automáticas de Uno a Muchos.

Comando	***Relación Uno a Muchos***
ADD RECORD	*Sí*
ADD SUBRECORD	*No*
APPLY TO SELECTION	*No*
DISPLAY SELECTION	*No*
EXPORT DIF	*No*
EXPORT SYLK	*No*
EXPORT TEXT	*No*
EXPORT DATA	*No*
MODIFY RECORD	*Yes*
MODIFY SUBRECORD	*No*
MODIFY SELECTION	*Sí (en entrada de datos)*
ORDER BY	*No*
ORDER BY FORMULA	*No*
QUERY BY FORMULA	*Sí*
QUERY SELECTION	*Sí*
QUERY	*Sí*
PRINT LABEL	*No*
PRINT SELECTION	*Sí*
QR REPORT	*No*
SELECTION TO Array	*No*
SELECTION RANGE TO Array	*No*

Activar por programación las relaciones entre tablas

Que las relaciones sean automáticas no significa que los registros relacionados o los registros para una tabla sean seleccionados simplemente porque un comando carga un registro. En algunos casos, después de ejecutar un comando que carga un registro, en ciertos casos usted debe llamar explícitamente los registros relacionados utilizando ***RELATE ONE*** *o* ***RELATE MANY*** *si necesita acceder a los datos relacionados.*

Algunos de los comandos listados en la tabla anterior (tal como los comandos de búsqueda) cargan el registro actual una vez se termina la tarea. En este caso, el registro cargado no selecciona automáticamente los registros relacionados a él. Nuevamente, si necesita acceder a los datos relacionados, debe seleccionar explícitamente los registros seleccionados utilizando ***RELATE ONE*** *o* ***RELATE MANY****.*

CREATE RELATED ONE

CREATE RELATED ONE (campo)

Parámetro	***Tipo***		***Descripción***
campo	*Campo*	➡	*Campo Muchos*

Descripción

CREATE RELATED ONE *tiene dos acciones. Si no existe un registro relacionado a campo (es decir, si el valor actual de campo, no se encuentra en el campo correspondiente de ningún registro de la tabla relacionada),* ***CREATE RELATED ONE*** *crea un nuevo registro relacionado.*

Para guardar en este registro el valor del campo que ha provocado su creación, asígnelo al campo correspondiente. Utilice el comando ***SAVE RELATED ONE*** *para guardar el nuevo registro.*

Si ya existe un registro relacionado, ***CREATE RELATED ONE*** *actúa como* ***RELATE ONE*** *y carga el registro relacionado en memoria.*

GET AUTOMATIC RELATIONS

GET AUTOMATIC RELATIONS (uno ; muchos)

Parámetro	Tipo		Descripción
uno	Booleano	⇐	Estado de todas las relaciones de Muchos a Uno
muchos	Booleano	⇐	Estado de todas las relaciones de Uno a Muchos

Descripción

*El comando **GET AUTOMATIC RELATIONS** le permite saber si el estado automático/manual de todas las relaciones manuales Muchos a Uno y Uno a Muchos de la base han sido modificadas en el proceso actual.*

- *uno: este parámetro devuelve **True** si una llamada anterior al comando **SET AUTOMATIC RELATIONS** vuelve automáticas todas las relaciones manuales Muchos a Uno, por ejemplo **SET AUTOMATIC RELATIONS(True;False)**.*

*Este parámetro devuelve **False** si el comando **SET AUTOMATIC RELATIONS** no ha sido llamado o si su ejecución previa no modificó las relaciones manuales Muchos a Uno, por ejemplo **SET AUTOMATIC RELATIONS(False;False)**.*

- *muchos: este parámetro devuelve **True** si la llamada previa al comando **SET AUTOMATIC RELATIONS** vuelve automáticas todas las relaciones manuales Uno a Muchos, por ejemplo **SET AUTOMATIC RELATIONS(True;True)**.*

*Este parámetro devuelve **False** si el comando **SET AUTOMATIC RELATIONS** no ha sido llamado o si si ejecución previa no modificó la relaciones manuales Uno a Muchos, por ejemplo **SET AUTOMATIC RELATIONS(True;False)**.*

Ejemplo

*Consulte el ejemplo del comando **GET FIELD RELATION**.*

GET FIELD RELATION

GET FIELD RELATION (campoN ; uno ; muchos {; *})

Parámetro	Tipo		Descripción
campoN	Campo	➡	Campo de inicio de una relación
uno	Entero largo	⬅	Estado de la relación Muchos a Uno
muchos	Entero largo	⬅	Estado de la relación Uno a Muchos
*	Operador	➡	• Si se pasa: uno y muchos devuelven el estado actual de la relación (valores 2 o 3 únicamente) • Si se omite (por defecto): uno y muchos puede devolver el valor 1 si la relación no ha sido modificada por programación

Descripción

*El comando **GET FIELD RELATION** permite conocer el estado automático/manual de la relación comenzando desde el campoN para el proceso actual. Todas las relaciones pueden ser consultadas, incluyendo las relaciones automáticas en la ventana de Estructura.*

- *Pase en campoN, el nombre del campo de la tabla N desde donde comienza la relación cuyo estado quiere conocer. Si ninguna relación se origina desde el campo campoN, los parámetros uno y muchos devuelven 0, se genera un error y la variable sistema OK toma el valor 0 (ver a continuación).*
- *Después de la ejecución del comando, la variable uno contiene un valor indicando si la relación Muchos a Uno especificada está definida como automática:*
  0 = no hay relación desde campoN. Se genera el error de sintaxis No. 16 ("El campo no tiene relación") y la variable sistema OK toma el valor 0.
  *1 = el estado automático/manual de la relación Muchos a Uno especificado está definido por la opción **Relación Muchos a Uno automática** en las propiedades de la relación en el entorno Diseño (no ha sido modificado por programación).*
  2 = la relación Muchos a Uno es manual para el proceso.
  3 = la relación Muchos a Uno es automática para el proceso.

- *Después de la ejecución del comando, el parámetro muchos contiene un valor indicando si la relación Uno a Muchos especificada está definida como automática:*
  0 = no hay relación desde campoN. Se genera el error de sintaxis No. 16 ("El campo no tiene relación") y la variable sistema OK toma el valor 0.
  *1 = el estado automático/manual de la relación Uno a Muchos especificada está definido por la opción **Relación Uno a Muchos automática** en las propiedades de la relación en el entorno Diseño (no ha sido modificado por programación).*
  2 = la relación Uno a Muchos es manual para el proceso.
  3 = la relación Uno a Muchos es automática para el proceso.

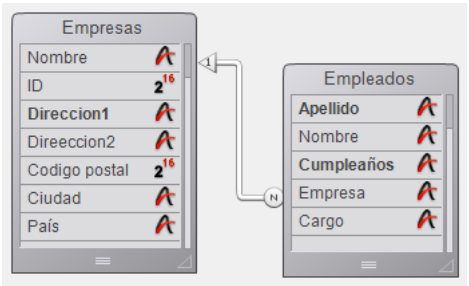
Puede comparar los valores recibidos en los parámetros uno y muchos con las constantes del tema "":

Constante	Tipo	Valor
Automatic	Entero largo	3
Manual	Entero largo	2
No relation	Entero largo	0
Structure configuration	Entero largo	1

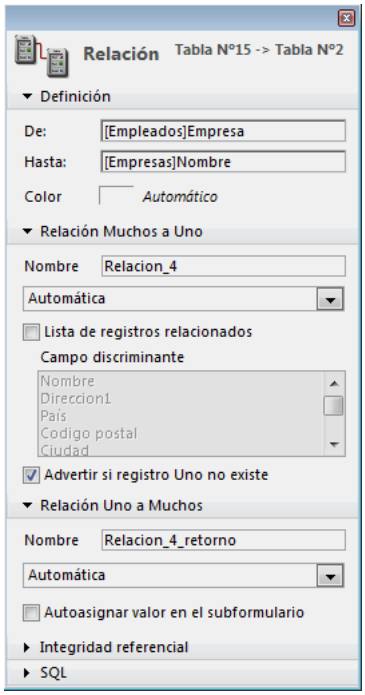
- *El parámetro opcional * permite "forzar" la lectura del estado actual de la relación, incluso si no ha sido modificado por programación. En otras palabras, cuando pasa el parámetro opcional *, sólo pueden devolverse los valores 2 ó 3 en los parámetros uno y muchos.*

Ejemplo

Dada la siguiente estructura:



Las propiedades de la relación que une el campo [Empleados]Empresa con el campo[Empresas]Nombre son las siguientes:



*El siguiente código ilustra las diferentes posibilidades ofrecidas por los comandos **GET FIELD RELATION**, **GET AUTOMATIC RELATIONS** y **SET FIELD RELATION**, **SET AUTOMATIC RELATIONS** así como sus efectos:*

```
GET AUTOMATIC RELATIONS(one;many)  `devuelve False, False
GET FIELD RELATION([Empleados]Empresa;one;many)  `devuelve 1,1
GET FIELD RELATION([Empleados]Empresa;one;many;*)  `devuelve 3,2

SET FIELD RELATION([Empleados]Empresa;2;0)  `cambia la relación Muchos a uno a manual

GET FIELD RELATION([Empleados]Empresa;one;many)  `devuelve 2,1
GET FIELD RELATION([Empleados]Empresa;one;many;*)  `devuelve 2, 2

SET FIELD RELATION([Empleados]Empresa;1;0)  `restablece los parámetros definidos en el entorno Diseño para la relación Muchos a Uno

GET FIELD RELATION([Empleados]Empresa;one;many)  `devuelve 1,1
GET FIELD RELATION([Empleados]Empresa;one;many;*)  `devuelve 3,2

SET AUTOMATIC RELATIONS(True;True)  `cambia todas las relaciones de todas las tablas a automática

GET AUTOMATIC RELATIONS(one;many)  `devuelve True, True
GET FIELD RELATION([Employees]Company;one;many)  `devuelve 1,1
GET FIELD RELATION([Employees]Company;one;many;*)  `devuelve 3,3
```

OLD RELATED MANY

OLD RELATED MANY (campo)

Parámetro	***Tipo***		***Descripción***
campo	*Campo*	➡	*Campo Uno*

Descripción

OLD RELATED MANY *funciona de la misma forma que* ***RELATE MANY****, excepto que* ***OLD RELATED MANY*** *utiliza el valor anterior del campo Uno para establecer la relación.*

Nota: ***OLD RELATED MANY*** *utiliza el valor anterior del campo Muchos, tal como lo devuelve la función* ***Old****. Para mayor información, consulte la descripción del comando* ***Old****.*

OLD RELATED MANY *cambia la selección de la tabla relacionada y selecciona el primer registro de la selección actual como registro actual.*

OLD RELATED ONE

OLD RELATED ONE (unCampo)

Parámetro	Tipo		Descripción
unCampo	Campo	➡	Campo Muchos

Descripción

OLD RELATED ONE funciona de la misma forma que **RELATE ONE**, con la diferencia de que **OLD RELATED ONE** utiliza el valor anterior de campo para establecer la relación.

Nota: OLD RELATED ONE utiliza el valor anterior del campo Muchos devuelto por la función **Old**. Para mayor información, consulte la descripción del comando **Old**.

OLD RELATED ONE carga el registro anteriormente relacionado al registro actual. Entonces puede accederse a los campos de este registro. Si quiere modificar este registro relacionado anterior y guardarlo, debe llamar a **SAVE RELATED ONE**. Tenga en cuenta que un registro que acaba de ser creado, no tiene un registro relacionado anterior.

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente y si los registros relacionados se han cargado, la variable sistema OK toma el valor 1. Si el usuario hizo clic en **Cancelar** en la caja de diálogo de selección del registro (que aparece cuando el registro relacionado ha sido modificado), la variable OKtoma el valor 0.

RELATE MANY

RELATE MANY (tabla1 | campo1)

Parámetro	Tipo		Descripción
tabla1 | campo1	Tabla, Campo	➡	Tabla para la cual establecer todas las relaciones de Uno a Muchos o campo Uno

Descripción

RELATE MANY *tiene dos sintaxis.*

La primera sintaxis, ***RELATE MANY(tabla1)****, establece todas las relaciones Uno a Muchos para tabla1. Cambia la selección actual para cada tabla que tiene una relación Uno a Muchos con tabla1. Las selecciones actuales en las tablas Muchos dependen del valor actual de cada campo relacionado en la tabla Uno. Cada vez que este comando se ejecuta, las selecciones actuales de las tablas Muchos son modificadas.*

La segunda sintaxis, ***RELATE MANY(campo1)****, establece la relación Uno a Muchos para campo1. Modifica la selección actual para cada tabla que tenga una relación con campo1. Esto significa que los registros relacionados se vuelven la selección actual de la tabla Muchos.*

Nota: *si la selección actual de la tabla Uno está vacía en el momento de la ejecución de* ***RELATE MANY****, el comando no hace nada.*

Nota: *este comando no soporta campos de tipo Objeto.*

Ejemplo

En el siguiente ejemplo, tres tablas están relacionadas con relaciones automáticas. Las dos tablas [Personas] y [Partes] tienen una relación Muchos a Uno con la tabla [Empresas].

Este es el formulario para la tabla [Empresas] que mostrará los registros relacionados de las tablas [Personas] y [Partes].

Cuando se muestran los formularios para Personas y Partes, los registros relacionados para las tablas [Personas] y [Partes] se cargan y se vuelven las selecciones actuales de estas tablas.

Por otra parte, los registros relacionados no se cargan si un registro de la tabla [Empresas] es seleccionado por programación. En este caso, debe utilizar el comando ***RELATE MANY****.*

Notas:

• Cuando el comando ***RELATE MANY*** *se aplica a una selección vacía, el comando no se ejecuta y la selección para la tabla Muchos no cambia.*

• Para que el comando funcione, los campos llave foránea (campos Muchos) deben estar indexados.

Por ejemplo, el siguiente método efectúa un bucle por cada registro de la tabla [Empresas]. Para cada empresa, aparece una caja de alerta. La caja de alerta muestra el número de personas en la empresa (el número de registros relacionados en la tabla [Personas]), y el número de partes que suministran (el número de registros en la tabla [Partes] que están relacionados). En el ejemplo, el argumento para el comando ***ALERT*** *se imprime en varias líneas por claridad.*

Note que es necesario el comando ***RELATE MANY****, aunque las relaciones sean automáticas.*

```
ALL RECORDS([Empresas]) ` Seleccionar todos los registros en la tabla
ORDER BY([Empresas];[Empresas]Nombre) ` Ordenar los registros en orden alfabético
For($i;1;Records in table([Empresas])) ` Bucle una vez por cada registro
   RELATE MANY([Empresas]Nombre) ` Seleccionar los registros relacionados
   ALERT("Company: "+[Empresas]Nombre+Char(13)+"Personas en la empresa: "
   +String(Records in selection([Personas]))+Char(13)+
   "Número de partes que suministran: "+String(Records in selection([Partes])))
   NEXT RECORD([Empresas]) ` Ir al siguiente registro
End for
```

RELATE MANY SELECTION

RELATE MANY SELECTION (unCampo)

Parámetro	Tipo		Descripción
unCampo	Campo	➡	Campo de la tabla Muchos (donde inicia la relación)

Descripción

*El comando **RELATE MANY SELECTION** crea una selección de registros en la tabla Muchos, basada en la selección actual de la tabla Uno.*

***Nota: RELATE MANY SELECTION** cambia el registro actual de la tabla Uno.*

Ejemplo

Este ejemplo selecciona todas las facturas de clientes con crédito superior o igual a $1 000. El campo [Facturas]IDCliente está relacionado con el campo [Clientes]NumID.

```
` Seleccionar los clientes
QUERY([Clientes];[Clientes]Credit>=1000)
` Buscar todas las facturas relacionadas con cada uno de estos clientes
RELATE MANY SELECTION([Facturas]IDCliente)
```

RELATE ONE

RELATE ONE (tablasN | CampoN {; discriminante})

Parámetro	Tipo		Descripción
tablasN | CampoN	Tabla, Campo	➡	Tabla para la cual definir todas las relaciones automáticas o Campo con la relación manual con la tabla Uno
discriminante	Campo	➡	Campo discriminante de la tabla 1

Descripción

RELATE ONE *acepta dos sintaxis.*

La primera sintaxis del comando, ***RELATE ONE****(tablaN), activa todas las relaciones Muchos a Uno automáticas para la tabla tablaN en el proceso actual. Esto significa que para cada campo de la tabla tablaN que tenga una relación Muchos a Uno automática, el comando seleccionará el registro relacionado en cada tabla relacionada. Esto cambia el registro actual en la(s) tabla(s) relacionadas para el proceso.*

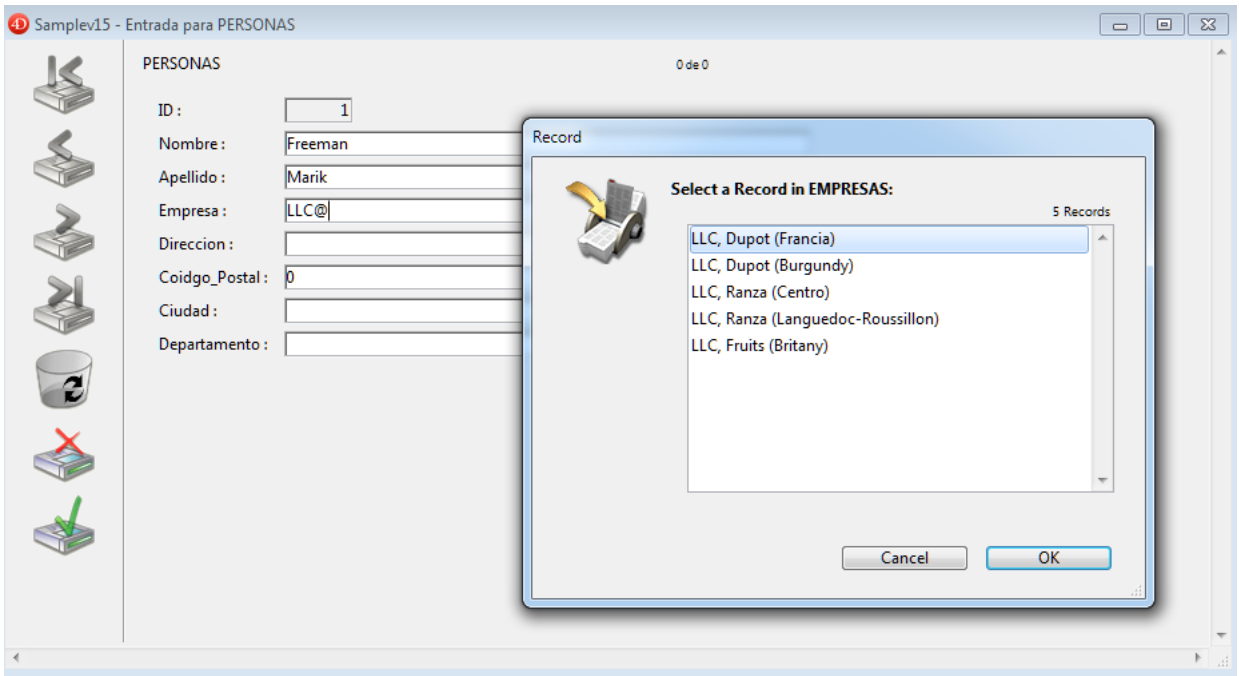
La segunda sintaxis, ***RELATE ONE****(campoN{;discriminante}), busca el registro relacionado con campoN. No es necesario que la relación sea automática. Si existe,* ***RELATE ONE*** *carga en memoria el registro relacionado, volviéndolo el registro y la selección actual de la tabla.*

El parámetro opcional discriminante debe ser un campo de la tabla relacionada. Puede ser únicamente de tipo Alfa, Texto, numérico, Fecha, Hora o Booleano. En particular, no puede ser tipo Imagen o BLOB.

Si campoN se especifica y si más de un registro se encuentra en la tabla relacionada, ***RELATE ONE*** *muestra una lista de registros que corresponden al valor de campoN, permitiendo al usuario seleccionar un registro. En esta lista, la columna de la izquierda muestra los valores de los campos relacionados, y la columna de la derecha los valores de discriminante.*

Se podría encontrar más de un registro si campoN termina con el carácter arroba (@). Si sólo hay una coincidencia, no aparece la lista.

En la pantalla abajo, se está introduciendo y se muestra una lista de selección en el primer plano.



El siguiente comando se utiliza para que aparezca la lista de selección:

```
RELATE ONE([Personnel]Company;[Companies]Region)
```

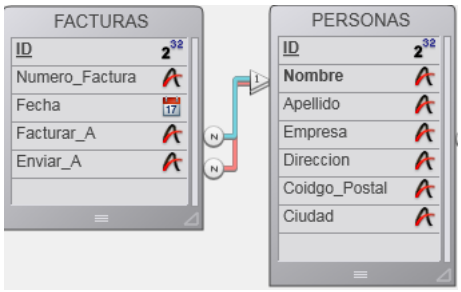
Un usuario introducido LLC@ para ver la lista de todas las empresas cuyos nombres comienzan por LLC, así como también su región.

Especificar un campo en discriminante es lo mismo que definir un campo discriminante en la caja de diálogo de definición de las propiedades de una relación en el entorno Diseño. Para mayor información sobre la definición de un campo discriminante, consulte el Manual de Diseño de 4D.

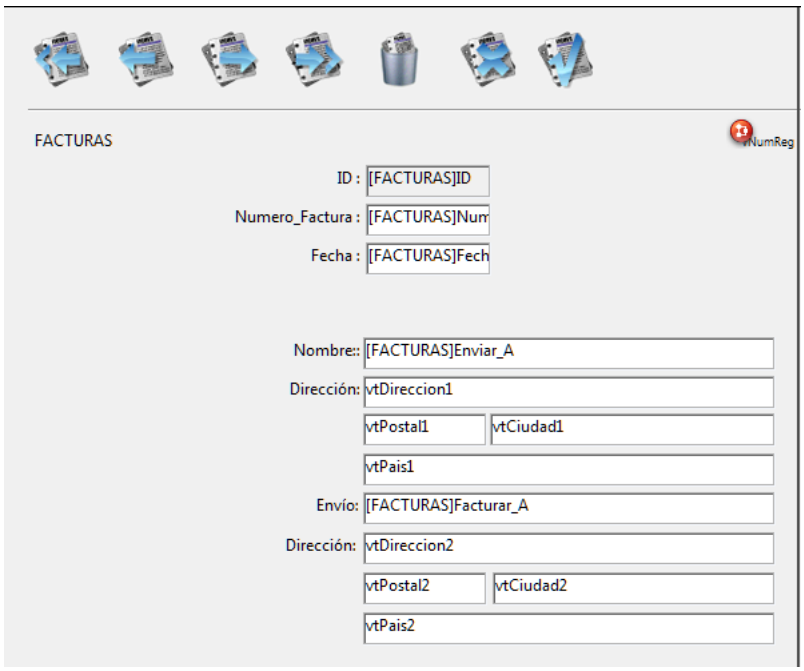
Nota: *este comando no soporta campos de tipo Objeto.*

Ejemplo

En el siguiente ejemplo, la tabla [Facturas] está relacionada a la tabla [Clientes] por dos relaciones manuales. Una relación parte del campo [Facturas]A nombre de al campo [Clientes]ID y la otra relación va de [Facturas]Enviar a a [Clientes]ID.



Esta es el formulario de la tabla [Invoices] que muestra la información "Facturar a" y "Enviar a":



Como las dos relaciones apuntan a la misma tabla, [Clientes], no es posible obtener la información de facturación y envío al mismo tiempo. Por lo tanto, la información debe ser mostrada utilizando variables y llama a ***RELATE ONE****. Si los campos [Clientes] se mostraron en su lugar, sólo se mostrarán los datos de una de las relaciones.*

Los siguientes métodos son los métodos de objeto de los campos [Facturas]Facturar a y [Facturas]Enviar a. Estos métodos se ejecutan cuando se introducen los campos.

Este es el método de objeto para el campo [Facturas]Facturar a:

```
RELATE ONE([Facturas]Enviar a)
vAddress1:=[Clientes]Direccion
vCity1:=[Clientes]Ciudad
vState1:=[Clientes]Estado
vZIP1:=[Clientes]CodigoPostal
```

Este es el método de objeto para el campo [Facturas]Enviar a:

```
RELATE ONE([Facturas]Enviar a)
vAddress2:=[Clientes]Direccion
vCity2:=[Clientes]Ciudad
vState2:=[Clientes]Estado
vZIP2:=[Clientes]CodigoPostal
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente y si los registros relacionados se han cargado, la variable sistema OK toma el valor 1. Si el usuario hizo clic en ***Cancelar*** *en la caja de diálogo de selección del registro (que aparece cuando el registro relacionado ha sido modificado), la variable* ***OK*** *toma el valor 0.*

RELATE ONE SELECTION

RELATE ONE SELECTION (tablaN ; tabla1)

Parámetro	Tipo		Descripción
tablaN	Tabla	➡	Nombre de la tabla Muchos (donde inicia la relación)
tabla1	Tabla	➡	Nombre de la tabla Uno (donde termina la relación)

Descripción

*El comando **RELATE ONE SELECTION** crea una nueva selección de registros en tabla1 a partir de la selección de registros de la tablaN relacionada y carga el primer registro de la nueva selección como el registro actual.*

*Este comando sólo puede utilizarse si hay una relación de Muchos a Uno. **RELATE ONE SELECTION** puede funcionar a través de varios niveles de relaciones. Puede haber varias tablas relacionadas entre la tabla Muchos y la tabla 1. Las relaciones pueden ser manuales o automáticas.*

***RELATE ONE SELECTION** utiliza la "ruta más corta" para pasar de la taba de inicio a la tabla de destino. Si las rutas existentes son de tamaño equivalente, debe llamar al comando **SET FIELD RELATION** si quiere controlar la ruta utilizada.*

Ejemplo

El siguiente ejemplo busca los clientes cuyas facturas se vencen hoy.

Esta es una manera de crear una selección en la tabla [Clientes], a partir de una selección de registros de la tabla [Facturas]:

```
CREATE EMPTY SET([Clientes];"Fecha de vencimiento del pago")
QUERY([Facturas];[Facturas]FechaVence=Current date)
While(Not(End selection([Facturas])))
   RELATE ONE([Facturas]CustID)
   ADD TO SET([Clientes];"Fecha de vencimiento del pago")
   NEXT RECORD([Invoices])
End while
```

*La siguiente técnica utiliza **RELATE ONE SELECTION** para obtener el mismo resultado:*

```
QUERY([Facturas];[Facturas]FechaVence=Current date)
RELATE ONE SELECTION([Facturas];[Clientes])
```

***Note:** a partir de la versión 11, este código puede escribirse de esta manera sin perdida de rendimiento:*

```
QUERY([Clientes];[Facturas]FechaVence=Current date)
```

SAVE RELATED ONE

SAVE RELATED ONE (unCampo)

Parámetro	***Tipo***		***Descripción***
unCampo	*Campo*	➡	*Campo Muchos*

Descripción

SAVE RELATED ONE *guarda el registro relacionado a campo.Ejecute este comando para actualizar un registro creado con* ***CREATE RELATED ONE****, o para guardar los cambios realizados a un registro cargado por* ***RELATE ONE****.*

SAVE RELATED ONE *no guardará un registro bloqueado. Cuando utilice este comando, primero debe asegurarse de que el registro no esté bloqueado. Si el registro está bloqueado, se ignora el comando, no se guarda el registro y no se devuelve ningún error.*

SET AUTOMATIC RELATIONS

SET AUTOMATIC RELATIONS (uno {; muchos})

Parámetro	Tipo		Descripción
uno	Booleano	➡	Estado de todas las relaciones Muchos a Uno
muchos	Booleano	➡	Estado de todas las relaciones de Uno a Muchos

Descripción

SET AUTOMATIC RELATIONS *cambia temporalmente todas las relaciones manuales en relaciones automáticas para toda la base en el proceso actual. Las relaciones permanecen automáticas a menos que se realice una llamada posterior a* ***SET AUTOMATIC RELATIONS****.*

- *Si uno es true, entonces todas las relaciones Muchos a Uno se vuelven automáticas. Si uno es false, todas las relaciones Muchos a Uno se vuelven manuales.*
- *Si muchos es true, entonces todas las relaciones Uno a Muchos se vuelven automáticas. Si muchos es false, todas las relaciones Uno a Muchos se vuelven manuales.*

Este comando cambia relaciones definidas como manuales en modo Diseño a automáticas, justo antes de ejecutar operaciones que requieran que sean automáticas (tales como búsquedas relacionales y ordenaciones). Una vez termina la operación, las relaciones pueden cambiarse a manuales llamando nuevamente a ***SET AUTOMATIC RELATIONS****. La relaciones definidas como automáticas en el entorno Diseño no son afectadas por este comando.*

Nota: *cuando pase* ***True*** *al comando* ***SET AUTOMATIC RELATIONS****, el modo automático se "bloquea" para todas las relaciones manuales durante la sesión. En este caso, todas las llamadas al comando* ***SET FIELD RELATION*** *durante la misma sesión se ignoran, sin importar si estaban antes o después de* ***SET AUTOMATIC RELATIONS****. Para "desbloquear" el modo automático y tener en cuenta las llamadas a* ***SET FIELD RELATION****, pase* ***False*** *a* ***SET AUTOMATIC RELATIONS****.*

Ejemplo

El siguiente ejemplo vuelve automáticas todas las relaciones Muchos a Uno y restablece a manual todas las relaciones Uno a Muchos cambiadas previamente:

```
SET AUTOMATIC RELATIONS(True;False)
```

SET FIELD RELATION

SET FIELD RELATION (tablasN | CampoN ; uno ; muchos)

Parámetro	Tipo		Descripción
tablasN | CampoN	Tabla, Campo	➡	Tabla de inicio de las relaciones o Campo de inicio de la relación
uno	Entero largo	➡	Estado de la relación Muchos a Uno a partir del campo o de las relaciones Muchos a Uno de la tabla
muchos	Entero largo	➡	Estado de la relación Uno a Muchos a partir del campo o de las relaciones Uno a Muchos de la tabla

Descripción

*El comando **SET FIELD RELATION** permite definir separadamente el estado automático/manual de cada relación de la base para el proceso actual, sin tener en cuenta su estado inicial definido en el entorno Diseño en la ventana de propiedades de las relaciones.*

Pase en el primer parámetro, un nombre de tabla o campo:

- *Si pasa un nombre de campo (campoN), el comando se aplicará únicamente a la relación a partir del campo Muchos especificado.*
- *Si pasa un nombre de tabla (tablaN), el comando se aplicará a todas las relaciones a partir de la tabla Muchos especificada.*
- *Si no hay ninguna relación a partir del campo campoN o de la tabla tablaN, los parámetros uno y muchos devuelven 0, se genera el error de sintaxis No. 16 ("El campo no tiene relación") y la variable sistema OK toma el valor 0.*

*Pase en los parámetros uno y muchos, los valores que indican el cambio de estado automático/manual a aplicar respectivamente a la(s) relación(es) de tipo Muchos a Uno y Uno a Muchos. Puede utilizar las constantes del tema "**Relaciones**":*

- *Do not modifyDo not modify (0) = No modificar el estado actual de la(s) relación(es).*
- *Structure configuration (1) = Utilizar la configuración definida para la(s) relación(es) en la ventana de estructura de la aplicación.*
- *Manual (2) = Volver manual(es) la(s) relación(es) en el proceso actual.*
- *Automatic (3) = Volver automática(s) la(s) relación(es) en el proceso actual.*

***Nota:** los cambios realizados con este comando sólo aplican al proceso actual. La configuración de las relaciones definida utilizando las opciones de la ventana de propiedades de la relación no se modifica.*

***Note**: si pasa True al comando **SET AUTOMATIC RELATIONS** durante la misma sesión, las llamadas al comando SET FIELD RELATION se ignoran, sin importar si están ubicadas antes o después de **SET AUTOMATIC RELATIONS**. Para "bloquear" el modo automático y tener en cuenta llamadas a SET FIELD RELATION, pase False a **SET AUTOMATIC RELATIONS**.*

Ejemplo

Este comando simplifica la gestión de las relaciones con el editor de informes rápidos. En las versiones anteriores de 4D, era necesario pasar todas las relaciones a automáticas para poder utilizarlas en el editor. El siguiente código permite definir sólo las relaciones útiles como automáticas:

```
SET AUTOMATIC RELATIONS(False;False)  `Inicialización de las relaciones
  `Sólo se utilizarán las siguientes relaciones
SET FIELD RELATION([Facturas]ID_Cliente;Automatic;Automatic)
SET FIELD RELATION([Linea_Factura]ID_Factura;Automatic;Automatic)
QR REPORT([Facturas];Char(1);True;True;True)
```

Selecciones

- *ALL RECORDS*
- *APPLY TO SELECTION*
- *Before selection*
- *CREATE SELECTION FROM ARRAY*
- *DELETE SELECTION*
- *DISPLAY SELECTION*
- *Displayed line number*
- *End selection*
- *FIRST RECORD*
- *GET HIGHLIGHTED RECORDS*
- *GOTO SELECTED RECORD*
- *HIGHLIGHT RECORDS*
- *LAST RECORD*
- *MOBILE Return selection*
- *MODIFY SELECTION*
- *NEXT RECORD*
- *ONE RECORD SELECT*
- *PREVIOUS RECORD*
- *Records in selection*
- *REDUCE SELECTION*
- *SCAN INDEX*
- *Selected record number*
- *TRUNCATE TABLE*

ALL RECORDS

ALL RECORDS {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual seleccionar todos los registros o Tabla por defecto, si se omite

Descripción

ALL RECORDS *selecciona todos los registros de tabla para el proceso actual.* ***ALL RECORDS*** *hace del primer registro de la selección el registro actual y lo carga en memoria.* ***ALL RECORDS*** *devuelve los registros en el orden por defecto, que es el orden en el cual los registros son almacenados en el disco.*

Ejemplo

El siguiente ejemplo muestra todos los registros de la tabla [Personas]:

```
ALL RECORDS([Personas]) ` Selección de todos los registros en la tabla
DISPLAY SELECTION([Personas]) ` Mostrar los registros en el formulario de salida
```

APPLY TO SELECTION

APPLY TO SELECTION (tabla ; formula)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla en la cual aplicar la fórmula o Tabla por defecto, si se omite
formula	Instrucción	➡	Línea de código o método

Descripción

APPLY TO SELECTION aplica formula a cada registro de la selección actual de tabla. La formula puede ser una línea de instrucciones o un método. Si formula modifica un registro de tabla, el registro modificado se guarda. Si formula no modifica un registro, el registro no se guarda. Si la selección actual está vacía, **APPLY TO SELECTION** no tiene efecto. Si la relación es automática, la formula puede contener un campo de una tabla relacionada.

APPLY TO SELECTION puede ser utilizado para reunir información de la selección de registros (por ejemplo, cálculo de un total), o para modificar una selección (por ejemplo, pasando a mayúscula la primera letra de un campo). Si este comando se utiliza dentro de una transacción, todos los cambios pueden deshacerse si la transacción se cancela.

4D Server: el servidor no ejecuta ninguno de los comandos que se pasen en formula. Cada registro de la selección será enviado al equipo cliente para ser modificado.

Un termómetro de progresión se muestra mientras se ejecuta **APPLY TO SELECTION**. Para ocultarlo, utilice **MESSAGES OFF** antes de llamar a **APPLY TO SELECTION**. Si el termómetro de progreso se muestra, el usuario puede cancelar la operación.

Ejemplo 1

El siguiente ejemplo cambia a mayúsculas todos los nombres en la tabla [Empleados]:

```
APPLY TO SELECTION([Empleados];[Empleados]Apellido:=Uppercase([Empleados]Apellido))
```

Ejemplo 2

Si un registro está bloqueado durante la ejecución de **APPLY TO SELECTION** y ese registro se modifica, el registro no se guardará. Todos los registros bloqueados que se encuentren se colocan en un conjunto llamado **LockedSet**. Después de ejecutar **APPLY TO SELECTION**, es recomendable probar **LockedSet** para verificar si hay registros bloqueados. El siguiente bucle se ejecuta hasta que todos los registros hayan sido modificados:

```
Repeat
   APPLY TO SELECTION([Empleados];[Empleados]Apellido:=Uppercase([Empleados]Apellido))
   USE SET("LockedSet")  ` Selección de registros bloqueados únicamente
Until(Records in set("LockedSet")=0)  ` Hasta que no haya registros bloqueados
```

Ejemplo 3

Este ejemplo utiliza un método:

```
ALL RECORDS([Empleados])
APPLY TO SELECTION([Empleados];M_Cap)
```

Variables y conjuntos del sistema

Si el usuario hace clic en el botón Detener en el termómetro de progresión, la variable sistema OK toma el valor 0. De lo contrario, toma el valor 1.

Before selection

Before selection {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	→	Tabla para la cual probar si el puntero se encuentra antes del primer registro seleccionado o Tabla por defecto, si se omite
Resultado	Booleano	←	Sí (TRUE) o No (FALSE)

Descripción

Before selection *devuelve TRUE cuando el puntero del registro actual se encuentra antes del primer registro de la selección actual de tabla.* ***Before selection*** *generalmente es utilizado para verificar si el comando* ***PREVIOUS RECORD*** *ha movido el puntero del registro actual antes del primer registro. Si la selección actual está vacía,* ***Before selection*** *devuelve TRUE.*

Para mover el puntero del registro actual a la selección, utilice los comandos ***LAST RECORD****,* ***FIRST RECORD****, o* ***GOTO SELECTED RECORD****.* ***NEXT RECORD*** *no mueve el puntero a la selección.*

Before selection *devuelve TRUE en el encabezado cuando un informe está imprimiéndose con* ***PRINT SELECTION*** *o a partir del comando de menú Imprimir. Puede utilizar el siguiente código para probar el primer encabezado e imprimir un encabezado especial para la primera página:*

```
  ` Método de un formulario de salida utilizado para un informe
$vpFormTabla:=Current form table
Case of
  ` ...
   :(Form event=On Header)
  ` El área encabezado va a imprimirse
      Case of
         :(Before selection($vpFormTabla->))
  ` El código para la primera ruptura del encabezado va acá
  ` ...
      End case
End case
```

Ejemplo

Este formulario se utiliza durante la impresión de un informe. Define una variable vTitulo, a imprimir en el área del encabezado en la primera página:

```
  ` Método de formulario [Finanzas];"Array"
Case of
  ` ...
   :(Form event=On Header)
      Case of
         :(Before selection([Finanzas))
            vTitulo:="Informe de finanzas de 1997" ` Definir el título para la primera página
         Else
            vTitulo:="" ` Borrar el título para todas las otras páginas
      End case
End case
```

CREATE SELECTION FROM ARRAY

CREATE SELECTION FROM ARRAY (tabla ; regArray {; temp})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual crear la selección
regArray	Entero largo, Array booleano	➡	Array de números de registros, o Array de booleanos (True = el registro está en la selección, False = el registro no está en la selección)
temp	Cadena	➡	Nombre de la selección temporal a crear, o Aplicar el comando a la selección actual si el parámetro se omite

Descripción

El comando **CREATE SELECTION FROM ARRAY** crea la selección temporal temp a partir de:

- un array de números de registros absolutos regArray de tabla,
- o de un array de booleanos. En este caso, los valores del array indican la pertenencia (**True**) o no (**False**) de cada registro de tabla a la selección temp.

Si no pasa temp o si pasa una cadena vacía, el comando será aplicado a la selección actual, la cual será entonces actualizada.

Cuando usted utiliza un array de enteros largos con este comando, todos los números del array representan la lista de números de los registros en temp. Si un número es incorrecto (registro no creado), se genera el error -10503.

Nota: atención, debe asegurarse de que el array no contenga líneas que tengan el mismo valor, de lo contrario la selección resultante será incorrecta.

Cuando usted utiliza un array booleano con este comando, el elemento X del array indica si el registro número X es (**True**) o no (**False**) en temp. El número de elementos en regArray debe ser igual al número de registros en tabla. Si el array es más pequeño que el número de registros, sólo los registros definidos por el array pueden hacer parte de la selección.

Nota: con un array de booleanos, el comando utiliza elementos del número 0 al X-1.

Gestión de errores

Si un número de registro no es válido (registro no creado), se genera el error -10503. Puede interceptar este error con la ayuda de un método instalado por el comando **ON ERR CALL**.

DELETE SELECTION

DELETE SELECTION {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual borrar la selección actual o Tabla por defecto, si se omite

Descripción

DELETE SELECTION *borra la selección actual de registros de tabla. Si la selección actual está vacía,* ***DELETE SELECTION*** *no hace nada. Después de borrar los registros, la selección actual queda vacía. Los registros que se borran durante una transacción están bloqueados para los otros usuarios y procesos hasta que la transacción se valide o cancele.*

Advertencia: *la eliminación de una selección de registros es una operación definitiva y no puede deshacerse.*

Deseleccionar la opción ***Registros borrados definitivamente*** *en el Inspector de tablas le permite aumentar la velocidad de las eliminaciones durante el uso de* ***DELETE SELECTION*** *(ver* ***Registros borrados definitivamente*** *en el manual Modo Diseño).*

Ejemplo 1

El siguiente ejemplo muestra todos los registros de la tabla [Personas] y permite al usuario seleccionar cuáles borrar. El ejemplo tiene dos partes. La primera es un método para mostrar los registros. La segunda es un método de objeto para un botón Borrar. Este es el primer método:

```
ALL RECORDS([Personas]) ` Selección de todos los registros
FORM SET OUTPUT([Personas];"Lista") ` Definición del formulario para listar los registros
DISPLAY SELECTION([Personas]) ` Mostrar todos los registros
```

El siguiente es el método de objeto del botón Borrar, que aparece en el pie de página del formulario de salida. El método de objeto utiliza los registros seleccionados por el usuario (el conjunto sistema ***UserSet****) para borrar la selección. Note que si el usuario no selecciona ningún registro,* ***DELETE SELECTION*** *no tiene ningún efecto.*

```
  ` Solicitar confirmación de que el usuario realmente quiere borrar los registros
CONFIRM("Usted seleccionó"+String(Records in set("UserSet"))+" personas a borrar."
+Char(13)+"Haga clic en OK para borrarlas.")
If(OK=1)
   USE SET("UserSet") ` Utilizar los registros elegidos por el usuario
   DELETE SELECTION([Personas]) ` Borrar la selección de registros
End if
ALL RECORDS([Personas]) ` Selección de todos los registros
```

Ejemplo 2

Si se encuentra un registro bloqueado durante la ejecución de ***DELETE SELECTION****, ese registro no se borra. Todos los registros bloqueados se colocan en un conjunto sistema llamado* ***LockedSet****. Después de la ejecución de* ***DELETE SELECTION****, puede probar* ***LockedSet*** *para verificar si los registros estaban bloqueados. El siguiente bucle se ejecutará hasta que todos los registros se borren:*

```
Repeat ` Repetir para cada registro bloqueado
   DELETE SELECTION([EstaTabla])
   If(Records in set("LockedSet")#0) ` Si hay registros bloqueados
      USE SET("LockedSet") ` Seleccionar únicamente los registros bloqueados
   End if
Until(Records in set("LockedSet")=0) ` Hasta que no haya más registros bloqueados
```

DISPLAY SELECTION

DISPLAY SELECTION ({tabla}{; modoSelección}{; entradaList}{; *}{; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a mostrar, o Tabla por defecto, si se omite
modoSelección	Entero largo	➡	Modo de selección
entradaList	Booleano	➡	Autorizar entrada en lista
*		➡	Utilizar el formulario de salida en caso de selección de un solo registro y ocultar las barras de desplazamiento en el formulario de entrada
*		➡	Mostrar las barras de desplazamiento en el formulario de entrada (anular el segundo efecto del primer parámetro *)

Descripción

DISPLAY SELECTION muestra la selección de tabla, utilizando el formulario de salida. Los registros se muestran en una lista por la cual se puede desplazar similar a la del modo Diseño. Si el usuario hace doble clic en un registro, por defecto se muestra el registro en el formulario de entrada actual. La lista se muestra en la ventana del primer plano.

Para mostrar una selección y poder igualmente modificar un registro en el formulario de entrada actual después de hacer doble clic en él (como lo hace en la ventana del entorno Diseño), utilice **OBJECT SET ENTERABLE** en lugar de **DISPLAY SELECTION**.

La información a continuación aplica a ambos comandos, excepto por la información sobre la modificación de registros.

Después de ejecutar **DISPLAY SELECTION**, no hay registro actual. Utilice un comando tal como **FIRST RECORD** o **LAST RECORD** para seleccionar uno.

El parámetro modoSeleccion se utiliza para definir las posibilidades de selección de registros en la lista utilizando el ratón. En este parámetro puede pasar una de las siguientes constantes del tema "**Parámetro de formulario**":

Constante	Tipo	Valor	Comentario
Multiple selection	Entero largo	2	El usuario puede seleccionar varios registros al tiempo. Para seleccionar registros contiguos, haga clic en el primer registro a seleccionar, luego presione la tecla **Mayús** antes de hacer clic en el último registro a incluir en la selección. Para seleccionar registros no adyacentes, haga clic en cada registro por separado mientras presiona la tecla **Ctrl** (Windows) o **Comando** (Mac OS).
No selection	Entero largo	0	No es posible seleccionar un registro en la lista
Single selection	Entero largo	1	Sólo es posible seleccionar un registro a la vez

Si no pasa el parámetro modoSelección, por defecto se utiliza el modo "Selección múltiple".

El parámetro entradaLista le permite autorizar el modo "Entrada en lista" en la lista mostrada. Este modo permite al usuario seleccionar y modificar directamente los valores de los registros en el formulario de salida. Pase **True** para activar este modo o **False** para desactivarlo. Por defecto, si no pasa el parámetro entradaLista, el modo "Entrada en lista" se desactiva.

Recuerde que con el comando **DISPLAY SELECTION**, este parámetro sólo permite la selección de los valores en la lista y no su modificación. De hecho, el comando **DISPLAY SELECTION** carga los registros de la selección actual en modo sólo lectura. Sólo el comando **MODIFY SELECTION** permite efectivamente la entrada de valores.

Nota: el comando **OBJECT SET ENTERABLE** permite activar o desactivar fácilmente el modo Entrada en lista.

Algunas reglas relacionadas con el parámetro opcional *:

- Si la selección contiene sólo un registro y el primer parámetro opcional * no se utiliza, el registro se mostrará en el formulario de entrada en lugar del formulario de salida.
- Si el primer parámetro opcional * se especifica, el registro único será mostrado en el formulario de salida.
- Si el primer parámetro opcional * se especifica y el usuario muestra el registro en el formulario de entrada haciendo doble clic en él, se ocultarán las barras de desplazamiento del formulario. Para anular este efecto, pase el segundo parámetro opcional *.

Puede poner botones personalizados en el área del encabezado o del pie de página del formulario de salida para terminar la ejecución del comando **DISPLAY SELECTION**. Puede utilizar los botones automáticos Aceptar o Cancelar para salir, o utilizar un método de objeto que llame a los comandos **ACCEPT** o **CANCEL**. Cuando un formulario de salida llamado por el comando **DISPLAY SELECTION** no tiene botones, sólo la tecla **Escape** (Windows) o **Esc** (Mac OS) permiten salir de la lista.

Durante y después de la ejecución de **DISPLAY SELECTION**, los registros que el usuario selecciona se conservan en un conjunto llamado **UserSet**. **UserSet** está disponible por medio de **DISPLAY SELECTION** a los métodos de objeto de los botones, a los métodos llamados por los comandos de menú, así como para el método de proyecto que llamó **DISPLAY SELECTION**.

Ejemplo 1

El siguiente ejemplo selecciona todos los registros en la tabla [Personas]. El comando **DISPLAY SELECTION** muestra los registros y permite al usuario seleccionar los registros a imprimir. Finalmente, selecciona los registros con **USE SET**, y los imprime con **PRINT SELECTION**:

```
ALL RECORDS([Personas]) ` Selección de todos los registros
DISPLAY SELECTION([Personas];*) ` Visualización de los registros
USE SET("UserSet") ` Utilizar sólo los registros seleccionados por el usuario
PRINT SELECTION([Personas]) ` Imprimir los registros que el usuario seleccionó
```

Ejemplo 2

*Ver el ejemplo #6 del comando **Form event**. Este ejemplo muestra todas las pruebas que puede necesitar para efectuar monitoreo total de los eventos que ocurren durante la ejecución del comando **DISPLAY SELECTION**.*

Ejemplo 3

*Para reproducir las funcionalidades ofrecidas por el menú Registros del entorno Diseño cuando utiliza **DISPLAY SELECTION** o **MODIFY SELECTION** en modo Aplicación, proceda de la siguiente forma:*

a. En el entorno Diseño, cree una barra de menús con los comandos de menú que quiera, por ejemplo, Mostrar todos, Buscar y Ordenar.

*b. Asocie esta barra de menús (utilizando el menú "Barra de menús asociada" en la caja de diálogo de propiedades del formulario) con el formulario de salida utilizado con los comandos **DISPLAY SELECTION** o **MODIFY SELECTION**.*

c. Asocie los siguientes métodos de proyecto a sus comandos de menú:

```
` M_SHOW_ALL (asociado al comando de menú Mostrar todos)
$vpCurTabla:=Current form table
ALL RECORDS($vpCurTabla->)
```

```
` M_QUERY (asociado al comando de menú Buscar)
$vpCurTabla:=Current form table
QUERY($vpCurTable->)

` M_ORDER_BY (asociado al comando de menú Ordenar)
$vpCurTabla:=Current form table
ORDER BY($vpCurTabla->)
```

*También puede utilizar otros comandos, tales como **PRINT SELECTION**, **QR REPORT**, etc. para ofrecer todas las opciones de menú estándar que quiera cada vez que visualice o modifique una selección en el modo Aplicación. Gracias al comando **Current form table**, estos método son genéricos, y la barra de menús a los cuales soporta puede asociarse a todo formulario de salida de cualquier tabla.*

Displayed line number

Displayed line number -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo	➲	Número de línea mostrada

Descripción

*El comando **Displayed line number** funciona únicamente en el contexto del evento de formulario **On Display Detail.** Devuelve el número de la línea que está siendo procesada mientras una lista de registros es mostrada en pantalla. Si **Displayed line number** se llama cuando no se muestra una lista, devuelve 0.*

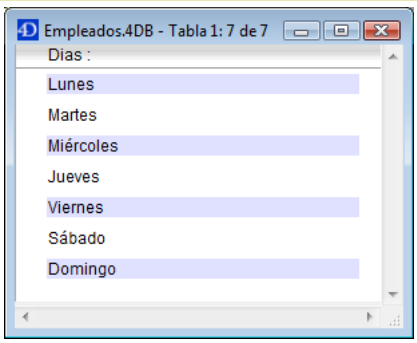
*Si la línea mostrada no está vacía (cuando está asociada a un registro), el valor devuelto por **Displayed line number** es idéntico al valor devuelto por **Selected record number**.*

*Como **Selected record number**, **Displayed line number** comienza en 1. Este comando es útil cuando quiere aplicar un proceso a cada línea de un formulario listado o de un list-box mostrado en pantalla, incluyendo las líneas vacías.*

Ejemplo

Este ejemplo le permite aplicar un color alterno en un formulario listado mostrado en pantalla, incluso para las líneas sin registros:

```
  `Método de formulario listado
If(Form event=On Display Detail)
   If(Displayed line number% 2=0)
  `Negro sobre blanco para el texto de las líneas pares
      OBJECT SET RGB COLORS([Tabla 1]Campo1;-1;0x00FFFFFF)
   Else
  `Negro sobre azul claro para las líneas impares
      OBJECT SET RGB COLORS([Tabla 1]Campo1;-1;0x00E0E0FF)
   End if
End if
```



End selection

End selection {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	→	Tabla en la cual probar si el puntero del registro está más allá del último registro seleccionado, o Tabla por defecto, si se omite
Resultado	Booleano	↩	Si (TRUE) o No (FALSE)

Descripción

End selection *devuelve TRUE cuando el puntero del registro actual está más allá del último registro de la selección actual de tabla.* ***End selection*** *se utiliza generalmente para probar si la llamada al comando* ***NEXT RECORD*** *ha movido o no el puntero del registro actual detrás del último registro de la selección. Si la selección actual está vacía,* ***End selection*** *devuelve TRUE.*

Para mover el puntero del registro actual dentro de la selección, utilice ***LAST RECORD***, ***FIRST RECORD***, *o* ***GOTO SELECTED RECORD***. ***PREVIOUS RECORD*** *no mueva el puntero a la selección.*

End selection *devuelve también TRUE cuando el último pie de página de un informe se imprime con* ***PRINT SELECTION*** *o desde el menú Imprimir. Puede utilizar el siguiente código para probar el último pie de de página e imprimir un pie de página especial para la última página:*

```
  ` Método de un formulario de salida utilizado para imprimir un informe
$vpFormTabla:=Current form table
Case of
  ` ...
   :(Form event=On Printing Footer)
  ` Se va a imprimir un pie de de página
      If(End selection($vpFormTabla->))
  ` El código para el último pie de página va aquí
      Else
  ` El código para el pie de página va aquí
      End if
End case
```

Ejemplo

Este método de formulario se utiliza durante la impresión de un informe. Se define la variable vPie a imprimir en el área de pie de página de la última página:

```
  ` [Finanzas];"Resumen" Método de formulario
Case of
  ` ...
   :(Form event=On Printing Footer)
      If(End selection([Finanzas]))
         vPie:="©2001 Acme Corp." ` Definir el pie de página de la última página
      Else
         vPie:="" ` Borrar el pie de página para todas las otras páginas
      End if
End case
```

FIRST RECORD

FIRST RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	⇒	Tabla de la cual cargar el primer registro de la selección actual o Tabla por defecto si se omite

Descripción

FIRST RECORD *hace del primer registro de la selección actual de tabla el registro actual y carga el registro del disco. Todos los comandos de búsqueda, selección, y ordenación también establecen como primer registro el registro actual. Si la selección actual está vacía o si el registro actual ya es el primer registro de la selección,* ***FIRST RECORD*** *no tiene ningún efecto.*

Este comando se utiliza con frecuencia después del comando ***USE SET*** *para iniciar un bucle en la selección de registros a partir del primer registro. Sin embargo, también puede llamarlo desde una subrutina si no está seguro de si el registro actual es realmente el primero.*

Ejemplo

El siguiente ejemplo carga el primer registro de la tabla [Clientes]:

```
FIRST RECORD([Clientes])
```

GET HIGHLIGHTED RECORDS

GET HIGHLIGHTED RECORDS ({tabla ;} nomConjunto)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual leer los registros seleccionados Si se omite, tabla del formulario actual
nomConjunto	Cadena	➡	Conjunto en el cual guardar los registros seleccionados

Descripción

El comando **GET HIGHLIGHTED RECORDS** guarda en el conjunto designado por el parámetro nomConjunto los registros seleccionados (es decir, los registros seleccionados por el usuario en el formulario listado) de la tabla pasada como parámetro. Si el parámetro tabla se omite, se utiliza la tabla del formulario o del subformulario actual.

En modo Diseño o durante la ejecución de los comandos **DISPLAY SELECTION** / **MODIFY SELECTION**, este comando puede remplazarse llamando el conjunto sistema UserSet mantenido automáticamente por 4D. Sin embargo, como este comando permite designar la tabla de la cual recuperar los registros seleccionados, el comando **GET HIGHLIGHTED RECORDS** también puede administrar las selecciones de registros en subformularios. En este caso, las selecciones de subformularios pueden provenir de diferentes tablas. Para mayor información sobre el conjunto UserSet, consulte la sección **Conjuntos**.

El comando **GET HIGHLIGHTED RECORDS** también puede llamarse fuera del contexto de un formulario; sin embargo, el conjunto devuelto está vacío.

El conjunto designado por nomConjunto puede ser local/cliente, proceso o interproceso.

Nota: en formularios incluidos, el comando **GET HIGHLIGHTED RECORDS** devuelve un conjunto vacío si el subformulario no tiene la propiedad de selección **Multilíneas**. En este caso, para conocer la línea seleccionada, debe utilizar el comando **Selected record number**.

Ejemplo

Este método indica cuántos registros están seleccionados en el subformulario que muestra los registros de la tabla [CDs]:

```
GET HIGHLIGHTED RECORDS([CDs];"$highlight")
ALERT(String(Records in set("$highlight"))+"registros seleccionados.")
CLEAR SET("$highlight")
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0.

GOTO SELECTED RECORD

GOTO SELECTED RECORD ({tabla ;} registro)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla en la cual ir al registro seleccionado o Tabla por defecto, si se omite
registro	Entero largo	➡	Posición del registro en la selección

Descripción

GOTO SELECTED RECORD vuelve el registro especificado en la selección actual de tabla el registro actual. La selección actual no cambia. El parámetro registro no es el mismo del número devuelto por **Record number**; Este parámetro representa la posición del registro en la selección actual. La posición del registro depende de la manera en que la selección ha sido creada y ordenada.

GOTO SELECTED RECORD no hace nada si:

- no hay registros en la selección actual
- registro no está en la selección actual,
- registro ya es el registro actual.

Si pasa 0 en registro, no habrá registro actual en tabla. Este mecanismo permite deseleccionar todos los registros en una lista, en particular en el caso de los subformularios incluidos, cuando el modo de selección es "único".

Ejemplo

El siguiente ejemplo carga datos del campo [Personas]Apellido en el array atNombres. Un array de enteros largos, llamado **NumReg**, se llena con los números que representarán a los registros seleccionados. Luego se ordenan los dos arrays:

```
` Crear aquí la selección de la tabla [Personas]
` ...
` Obtener los nombres
SELECTION TO ARRAY([Personas]Apellido;atNombres)
` Crear un array para los números de registros seleccionados
$vlNbRegistros:=Size of array(atNombres)
ARRAY LONGINT(NumReg;$vlNbRegistros)
For($vlRegistro ;1;$vlNbRegistros)
   NumReg{$vlRegistro }:=$vlRegistro
End for
` Ordenar los dos arrays en orden alfabético
SORT ARRAY(atNombres;NumReg;>)
```

Si el array atNombres se muestra en un área de desplazamiento, el usuario hace clic en uno de los elementos. Como la ordenación de los dos arrays está sincronizada, todo elemento de numReg proporciona el número del registro seleccionado para el registro cuyo nombre se guarda en el elemento correspondiente en atNombres.

El siguiente método de objeto del área desplegable atNombres selecciona el registro correcto en la selección de [Personas], de acuerdo al nombre elegido en el área de desplazamiento:

```
Case of
   :(Form event=On Clicked)
      If(atNombres#0)
         GOTO SELECTED RECORD(NumReg{atNombres})
      End if
End case
```

HIGHLIGHT RECORDS

HIGHLIGHT RECORDS ({tabla }{;}{ nomConjunto {; *}})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual seleccionar los registros Si se omite, tabla del formulario actual
nomConjunto	Cadena	➡	Conjunto de registros a seleccionar o Userset si se omite
*	Operador	➡	Desactivar el desplazamiento automático de la lista

Descripción

*El comando **HIGHLIGHT RECORDS** permite seleccionar registros en un formulario listado. Esta operación es idéntica a la selección de registros manual en modo listado utilizando el ratón o las combinaciones de teclado **Mayús+Clic** o **Ctrl+Clic** (Windows) o **comando+Clic** (Mac OS). La selección actual no se modifica.*

***Nota:** el conjunto de registros seleccionados se actualiza después de redibujar los registros; es decir, después de la ejecución de todo el método de llamado y no sólo inmediatamente después de la ejecución de **HIGHLIGHT RECORDS**.*

El parámetro tabla permite designar la tabla de la cual seleccionar los registros. Este parámetro se puede utilizar, particularmente, para seleccionar los registros de los subformularios incluidos, los cuales no pertenecen a la tabla actual (ver a continuación).

- *Si pasa un nombre de conjunto válido en nomConjunto, el comando se aplicará a los registros de ese conjunto para la tabla definida.*
- *Si omite el parámetro nomConjunto, el comando seleccionará únicamente los registros del conjunto UserSet actual. Este conjunto sólo se gestiona en modo Diseño y en caso de llamar los comandos **MODIFY SELECTION** / **DISPLAY SELECTION**. Si quiere seleccionar los registros de un subformulario, debe pasar un nombre de tabla y de conjunto. Para mayor información sobre el conjunto UserSet, consulte la sección .*

*Cuando se pasa el parámetro *, provoca la inactivación de la función de desplazamiento automático de la lista si los registros seleccionados no son visibles. Este mecanismo autoriza la gestión personalizada del desplazamiento vía el comando **OBJECT SET SCROLL POSITION**.*

***Nota:** en el marco de los subformularios incluidos, el comando **HIGHLIGHT RECORDS** no hace nada si la propiedad de selección **Multilíneas** no está seleccionada para el formulario. En este caso, para seleccionar una línea, debe utilizar el comando **GOTO SELECTED RECORD**.*

Ejemplo

*En un formulario de salida mostrado por el comando **MODIFY SELECTION**, usted quiere que el usuario pueda realizar búsquedas sin que la selección actual se modifique. Para hacer esto, coloque un botón **Buscar** en el formulario y asócielo con el siguiente método:*

```
SET QUERY DESTINATION(Into set;"UserSet")
QUERY
SET QUERY DESTINATION(Into current selection)
HIGHLIGHT RECORDS
```

Cuando el usuario hace clic en el botón, aparece la caja de diálogo estándar buscar. Una vez se valida la búsqueda, los registros encontrados se seleccionarán sin que la selección actual se modifique.

LAST RECORD

LAST RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual mover el último registro seleccionado o Tabla por defecto, si se omite

Descripción

LAST RECORD designa el último registro de la selección de tabla como registro actual y lo carga en memoria. Si la selección actual está vacía o si el registro actual ya es el último de la selección, **LAST RECORD** no tiene efecto.

Ejemplo

El siguiente ejemplo designa el último registro de la tabla [Personas] como registro actual:

```
LAST RECORD([Personas])
```

MOBILE Return selection

MOBILE Return selection (laTabla) -> Resultado

Parámetro	Tipo		Descripción
laTabla	Tabla	➡	Tabla de la cual retornar la selección actual
Resultado	Objeto	➲	Selección compatible Wakanda

Descripción

*El comando **MOBILE Return selection** devuelve un objeto JSON que contiene la selección actual de latabla transformada en una colección de entidades Wakanda.*

Este comando está destinado para ser llamado en el contexto de una conexión 4D Mobile, por lo general entre la aplicación 4D y una aplicación Wakand (vía REST). Cuando se establece una conexión 4D Mobile y se han configurado los derechos de acceso adecuados, una aplicación Wakanda puede ejecutar un método proyecto 4D que devuelve un valor en el parámetro $0.

*El comando **MOBILE Return selection** le permite devolver en $0, la selección actual de registros de la tabla laTabla, en la forma de un objeto tipo colección de entidades en formato JSON. Este objeto es compatible con las colecciones de entidades de Wakanda que contienen una selección de registros (es decir, de entidades).*

Tenga en cuenta que los accesos 4D Mobile requieren configuraciones específicas en sus bases 4D:

- *El servidor Web debe estar en marcha,*
- *La opción "Activar los servicios 4D Mobile" se estar seleccionada en las Propiedades de la base,*
- *Debe tener una licencia válida,*
- *Las tablas y campos utilizados deben tener seleccionada la opción "Expose para 4D Mobile" (seleccionada por defecto).*
- *Los métodos llamados deben tener la opción "Disponible vía las llamadas 4D Mobile" activa (no seleccionada por defecto).*

Tenga en cuenta que puede pasar toda tabla válida de la base en laTabla y no necesariamente la tabla con la que el método proyecto se ha asociado en sus propiedades. Este parámetro sólo se utiliza del lado Wakanda para definir los objetos para los que se puede llamar al método.

*Para obtener más información sobre la configuración 4D Mobile, consulte la documentación **4D Mobile**.*

Ejemplo

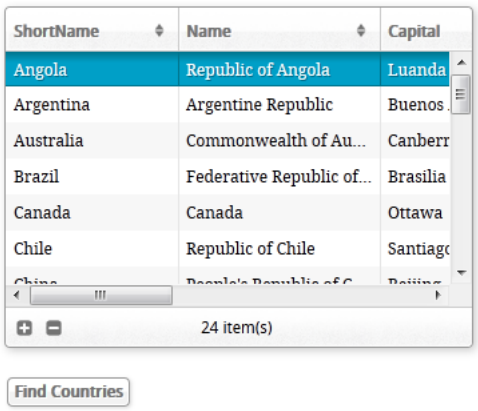
Usted desea mostrar la selección actual de la tabla [Countries] en una rejilla Wakanda, basado en una búsqueda.

Escribe el siguiente método:

```
//FindCountries project method
//FindCountries( string ) -> object

C_TEXT($1)
C_OBJECT($0)
QUERY([Countries];[Countries]ShortName=$1+"@")
$0:=MOBILE Return selection([Countries])
```

la selección devuelta puede ser utilizada directamente en Wakanda como una colección válida.

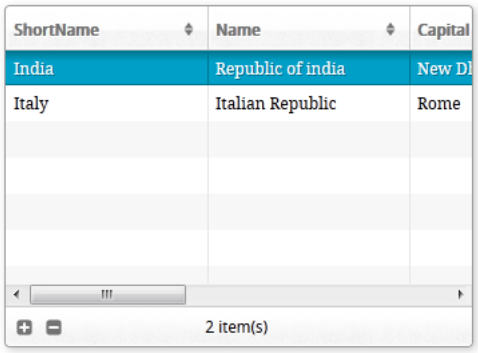
En el modelo del servidor de Wakanda conectado a 4D vía 4D Mobile, usted ha creado una página con una rejilla asociada a la tabla 4D Countries. Por defecto, durante la ejecución, se muestran todas las entidades de la tabla 4D:



El código del botón es:

```
button1.click = function button1_click (event) <p>      {           sources.countries.FindCountries("i", {  //llamamos al método 4D, "i" se pasa en $1
onSuccess:function(coll){  //función de retrollamada (asíncrona), recibe $0 como parámetro                    sources.countries.setEntityCollection(coll.result);
//remplaza la colección de entidades actual                                    // con la que recibe en el objeto coll.result                  }              });              };
```

Como resultado, la rejilla se actualiza:



MODIFY SELECTION

MODIFY SELECTION ({tabla}{; modoSelección}{; entradaList}{; *}{; *})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a mostrar y modificar o Tabla por defecto, si se omite
modoSelección	Entero largo	➡	Modo de selección
entradaList	Booleano	➡	Autorizar entrada en lista
*		➡	Utilizar formulario de salida para un sólo registro y ocultar las barras de desplazamiento en el formulario de entrada
*		➡	Mostrar las barras de desplazamiento en el formulario de entrada (anula la segunda opción del primer parámetro *)

Descripción

MODIFY SELECTION *es casi idéntico al comando* ***DISPLAY SELECTION****. Consulte la descripción del comando* ***DISPLAY SELECTION*** *para una descripción detallada. Las diferencias entre los dos comandos son:*

1. ***DISPLAY SELECTION*** *y* ***MODIFY SELECTION*** *permite mostrar los registros de la selección actual en modo listado o en el formulario de entrada al hacer doble clic en un registro. Utilizando* ***MODIFY SELECTION****, también puede modificar los campos del registro en el formulario de entrada cuando hace doble clic en el registro, (si no está siendo utilizado por otro proceso o usuario) o en modo "Entrada en lista" (si está autorizado).*

2. ***DISPLAY SELECTION*** *carga los registros en modo sólo lectura en el proceso actual, lo cual significa que no están bloqueados para escritura en los otros procesos.* ***MODIFY SELECTION*** *coloca todos los registros de la selección en modo lectura-escritura, lo que significa que son bloqueados automáticamente para escritura en otros procesos.* ***MODIFY SELECTION*** *libera los registros cuando termina su ejecución.*

NEXT RECORD

NEXT RECORD {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla en la cual mover el siguiente registro seleccionado o Tabla por defecto, si se omite

Descripción

NEXT RECORD mueve el puntero del registro actual al siguiente registro en la selección actual de tabla para el proceso actual. Si la selección actual está vacía, o si **Before selection** o **End selection** es TRUE, **NEXT RECORD** no tiene ningún efecto.

Si **NEXT RECORD** mueve el puntero del registro actual al final de la selección actual, **End selection** devuelve TRUE, y no hay registro actual. Si **End selection** devuelve TRUE, utilice **FIRST RECORD**, **LAST RECORD**, o **GOTO SELECTED RECORD** para mover el puntero del registro actual de regreso a la selección actual.

Ejemplo

Ver el ejemplo del comando **DISPLAY RECORD**.

ONE RECORD SELECT

ONE RECORD SELECT {(tabla)}

Parámetro	*Tipo*		*Descripción*
tabla	*Tabla*	➡	*Tabla en la cual reducir la selección actual al registro actual o Tabla por defecto si se omite*

Descripción

ONE RECORD SELECT *reduce la selección actual de tabla al registro actual. Si no existe un registro actual o si el registro actual no está cargado en la memoria (caso particular),* ***ONE RECORD SELECT*** *no tiene ningún efecto.*

Nota

Este comando fue útil para "reponer" un registro que había sido apilado y desapilado de la pila de registro mientras la selección de la tabla era modificada. ***SET QUERY DESTINATION*** *permite efectuar una búsqueda sin tener que cambiar la selección ni el registro actual de una tabla; por lo tanto, no necesita más apilar y desapilar un registro actual para efectuar una búsqueda en su tabla. Por consiguiente,* ***ONE RECORD SELECT*** *es menos útil, a menos que quiera expresamente reducir la selección de una tabla al registro actual.*

PREVIOUS RECORD

PREVIOUS RECORD {(tabla)}

Parámetro	*Tipo*		*Descripción*
tabla	*Tabla*	➡	*Tabla en la cual mover el registro anterior de la selección actual o Tabla por defecto, si se omite*

Descripción

PREVIOUS RECORD *mueve el puntero del registro actual al registro anterior en la selección de tabla para el proceso actual. Si la selección actual está vacía, o si* ***Before selection*** *o* ***End selection*** *es TRUE,* ***PREVIOUS RECORD*** *no tienen ningún efecto.*

Si ***PREVIOUS RECORD*** *mueve el puntero del registro actual antes de la selección actual,* ***Before selection*** *devuelve TRUE, y no hay registro actual. Si* ***Before selection*** *devuelve TRUE, utilice* ***FIRST RECORD***, ***LAST RECORD***, *o* ***GOTO SELECTED RECORD*** *para mover el puntero del registro actual en la selección actual.*

Records in selection

Records in selection {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual devolver el número de registros seleccionados,o tabla por defecto, si se omite
Resultado	Entero largo	➲	Registros en la selección de tabla

Descripción

Records in selection *devuelve el número de registros en la selección actual de tabla. En contraste,* ***Records in table*** *devuelve el número total de registros en la tabla.*

Ejemplo

El siguiente ejemplo muestra una técnica de bucle actualmente utilizada para moverse entre los registros de la selección actual. La misma operación puede realizarse con la ayuda del comando ***APPLY TO SELECTION****:*

```
FIRST RECORD([Personas])  ` Comenzar en el primer registro de la selección
For($vlRegistro;1;Records in selection([Personas]))  ` Bucle una vez por registro
   Hacer algo  ` Realizar una operación con el registro
   NEXT RECORD([Personas])  ` Pasar al siguiente registro
End for
```

REDUCE SELECTION

REDUCE SELECTION ({tabla ;} Numero)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual reducir la selección, o Tabla por defecto, si se omite
Numero	Entero largo	➡	Número de registros a conservar seleccionados

Descripción

REDUCE SELECTION *crea una nueva selección de registros para tabla. El comando devuelve el primer número de registros de la selección actual de tabla.* ***REDUCE SELECTION*** *se aplica a la selección actual de tabla en el proceso actual. El primer registro de la nueva selección actual es el registro actual.*

Nota: *si se ejecuta la instrucción REDUCE SELECTION(tabla;0), no hay más selección ni registro actual en tabla.*

Ejemplo

El siguiente ejemplo busca las estadísticas correctas para una competencia mundial entre los distribuidores de más de 20 países. Por cada país, los 3 mejores distribuidores que han vendido más de $50 000 de productos están entre los 100 mejores distribuidores en el mundo y reciben un premio. Con unas pocas líneas de código, esta petición compleja se puede ejecutar utilizando búsquedas indexadas:

```
CREATE EMPTY SET([Distribuidores];"Ganadores") ` Crear un conjunto vacío
SCAN INDEX([Distribuidores]Cantidad ventas;100;<) ` Buscar desde el final del índice
CREATE SET([Distribuidores];"100 mejores distribuidores") ` Colocar los registros seleccionados en un conjunto
For($Pais;1;Records in table([Paises])) ` Por cada país
  ` Buscar los distribuidores en este país
   QUERY([Distribuidores];[Distribuidores]País=[Países]Nombre;*) ` ...que vendieron más de $50 000
   QUERY(&;[Distribuidores];[Distribuidores]Cantidad ventas>=50000)
   CREATE SET([Distribuidores];"DistribuidoresGanadores") ` Colocarlos en un conjunto
  ` Deben estar en el gurpo de los 100 mejores distribuidores
   INTERSECTION("DistribuidoresGanadores";"100 mejores distribuidores";"DistribuidoresGanadores")
   USE SET("DistribuidoresGanadores") ` Ganadores potenciales por país
  ` Ordenarlos por los resultados en orden descendente
   ORDER BY([Distribuidores];[Distribuidores]Cantidad ventas;<)
   REDUCE SELECTION([Distribuidores];3) ` Tomar los tres mejores distribuidores
   CREATE SET([Distribuidores];"DistribuidoresGanadores") ` Ganadores por país
  ` Colocarlos en la lista de ganadores por país
   UNION("DistribuidoresGanadores";"Ganadores";"Ganadores")
End for
CLEAR SET("100 mejores distribuidores") `No necesitamos más este conjunto
CLEAR SET("DistribuidoresGanadores") ` No necesitamos más este conjunto
USE SET("Ganadores") ` Acá tiene los ganadores
CLEAR SET("Ganadores") ` No necesitamos más este conjunto
OUTPUT FORM([Distribuidores];"Carta de ganadores") ` Seleccionar la carta
PRINT SELECTION([Distribuidores]) ` Imprimir las cartas
```

SCAN INDEX

SCAN INDEX (unCampo ; Numero {; > o <})

Parámetro	Tipo		Descripción
unCampo	Campo	➡	Campo indexado con el cual escanear los registros
Numero	Entero largo	➡	Número de registros a devolver
> o <	Operador	➡	> a partir del inicio del índice < a partir del final del índice

Descripción

SCAN INDEX *devuelve una selección de numero de registros de la tabla. Si pasa <,* ***SCAN INDEX*** *devuelve el número de registros a partir del final del índice (valores superiores). Si pasa >,* ***SCAN INDEX*** *devuelve numero de registros a partir del inicio del índice (valores inferiores). Este comando es muy eficiente porque utiliza el índice para realizar la operación.*

Nota: *la selección que se obtiene no está ordenada.*

SCAN INDEX *funciona únicamente con campos indexados. Este comando modifica la selección actual de la tabla para el proceso actual, pero no hay registro actual.*

Si especifica un número de registros superior al número de registros presentes en la tabla, ***SCAN INDEX*** *devolverá todos los registros.*

Nota: *este comando no soporta campos de tipo Objeto.*

Ejemplo

El siguiente ejemplo envía cartas a los 50 peores clientes y 50 a los mejores clientes:

```
SCAN INDEX([Clientes]TotalVencido;50;<) ` Obtener la lista de los 50 peores clientes
ORDER BY([Clientes]CodigoPostal;>) ` Ordenar por código postal
FORM SET OUTPUT([Clientes];"Advertencia")
PRINT SELECTION([Clientes]) ` Imprimir las cartas
SCAN INDEX([Clientes]TotalVencido;50;>) ` Obtener la lista de los 50 mejores clientes
ORDER BY([Clientes]CodigoPostal;>) ` Ordenar por código postal
FORM SET OUTPUT([Clientes];"Carta de agradecimiento")
PRINT SELECTION([Clientes]) ` Imprimir las cartas
```

Selected record number

Selected record number {(tabla)} -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	→	Tabla de la cual devolver el número de registros seleccionados, o Tabla por defecto, si se omite
Resultado	Entero largo	↩	Número en la selección

Descripción

Selected record number *devuelve la posición del registro actual en la selección actual de tabla.*

Si la selección no está vacía y si el registro actual está en la selección, ***Selected record number*** *devuelve un valor entre 1 y* ***Records in selection****. Si la selección está vacía, o si no hay registro actual, devuelve 0 (cero).*

El número del registro en la selección es diferente del número devuelto por ***Record number****, que devuelve el número del registro físico en la tabla. El número del registro en la selección depende de la selección y el registro actual.*

Ejemplo

El siguiente ejemplo guarda el número del registro actual de la selección en una variable:

```
NumRegAc:=Selected record number([Personas])  ` Obtener el número del registro en la selección
```

TRUNCATE TABLE

TRUNCATE TABLE {(tabla)}

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual se borrarán todos los registros o tabla por defecto si se omite este parámetro

Descripción

*El comando **TRUNCATE TABLE** borra rápidamente todos los registros de tabla. Si tabla ya está vacía, TRUNCATE TABLE no hace nada. Después de llamar el comando, no hay una selección actual ni un registro actual.*

*El efecto de este comando es similar al de una secuencia **ALL RECORDS** / **DELETE SELECTION**; sin embargo, su funcionamiento difiere en los siguientes puntos:*

- *No se llama el trigger*
- *La integridad referencial de los datos no es controlada.*
- *Ninguna transacción debe estar en curso en el proceso que ejecuta **TRUNCATE TABLE**. Si este es el caso, el comando no hace nada y la variable sistema OK toma el valor 0*
- *Si uno o más registros están bloqueados por otro proceso, el comando falla: se genera un error y la variable sistema OK toma el valor 0. El conjunto sistema LockedSet no se crea.*
- *Si tabla ya está vacía, **TRUNCATE TABLE** no hace nada y la variable OK toma el valor 1.*
- *Si tabla está en modo sólo lectura, TRUNCATE TABLE no hace nada y la variable OK toma el valor 0.*
- *La variable OK toma el valor 0 ó 1 dependiendo de si el comando falló o fue exitoso.*
- *La operación se graba en el archivo de historial si lo hay.*

*El comando **TRUNCATE TABLE** debe por lo tanto utilizarse con precaución porque es muy efectivo en algunos casos, por ejemplo, para borrar rápidamente datos temporales.*

***Nota:** el concepto y funcionamiento de este comando es similar al del comando **SQL TRUNCATE** (TABLA).*

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0.

Selecciones temporales

Selecciones temporales
CLEAR NAMED SELECTION
COPY NAMED SELECTION
CUT NAMED SELECTION
USE NAMED SELECTION

Selecciones temporales

Las selecciones temporales ofrecen una manera fácil de manipular varias selecciones simultáneamente. Una selección temporal es una lista ordenada de registros para una tabla en un proceso. Esta lista ordenada puede tener un nombre y ser almacenada en memoria. Las selecciones temporales ofrecen una manera sencilla de guardar en memoria el orden de la selección y el registro actual de la selección.

Los siguientes comandos le permiten trabajar con selecciones temporales:

- ***COPY NAMED SELECTION***
- ***CUT NAMED SELECTION***
- ***USE NAMED SELECTION***
- ***CLEAR NAMED SELECTION***
- ***CREATE SELECTION FROM ARRAY***

*Las selecciones temporales se crean con los comandos **COPY NAMED SELECTION**, **CUT NAMED SELECTION** y **CREATE SELECTION FROM ARRAY**. Las selecciones temporales se utilizan generalmente para trabajar en una o más selecciones y para guardar y luego restaurar una selección ordenada. Puede haber varias selecciones temporales para cada tabla en un proceso. Para reutilizar una selección temporal como selección actual, llame **USE NAMED SELECTION**. Cuando haya terminado de utilizar una selección, utilice **CLEAR NAMED SELECTION**.*

***Nota:** La combinación de la instrucción **SET QUERY DESTINATION(Into named selection;namedselection)** con un comando de búsqueda (por ejemplo **QUERY**) también puede utilizarse para crear una selección temporal. Consulte la descripción del comando **SET QUERY DESTINATION**.*

Las selecciones temporales pueden tener un alcance proceso o interproceso.

Una selección temporal es interproceso si su nombre está precedido por los símbolos (<>), un signo "menor que" seguido por un signo "mayor que".

***Nota**: esta sintaxis puede utilizarse en Windows y Macintosh. Además, en Macintosh, puede utilizar el símbolo diamante (Opción-Mayús-V en teclado en español).*

El alcance de una selección temporal interproceso es idéntico al alcance de una variable interproceso. Se puede acceder a una selección temporal interproceso desde cualquier proceso.

Una selección temporal cuyo nombre no tiene como prefijo los símbolos (<>) es proceso en alcance y está disponible sólo en el proceso en el cual fue creada.

Con 4D en modo remoto y 4D Server, una selección temporal interproceso está disponible únicamente para los procesos del cliente que la creó. Una selección temporal interproceso no está disponible en otros equipos clientes.

Una selección temporal proceso sólo está disponible dentro del proceso en el cual fue creada y en el servidor.

Una selección temporal local está definida por el proceso que la creó y no es visible en el servidor.

***Advertencia**: la creación de una selección temporal necesita acceso a la selección de la tabla. Como las selecciones se mantienen en el servidor y un proceso local no tiene acceso al servidor de datos, no utilice selecciones temporales en procesos locales.*

Visibilidad de Conjuntos

La siguiente tabla indica los principios de visibilidad de los conjuntos en función de su alcance y de dónde fueron creados:

	Procesos Cliente	Otros Procesos Cliente	Otros Clientes	Procesos Servidor	Otros Procesos Servidor
Creación de un proceso cliente					
$prueba	x				
prueba	x			x(Trigger)	
<>prueba	x	x			
Creación de un proceso servidor					
$prueba				x	
prueba				x	
<>prueba				x	x

Selecciones temporales y conjuntos

Las diferencias entre conjuntos y selecciones temporales son:

- *Una selección temporal es una lista ordenada de registros; un conjunto no.*
- *Los conjuntos son eficientes en el uso de memoria, porque necesitan sólo un bit por cada registro en el archivo. Las selecciones temporales necesitan 4 bytes para cada registro en la selección.*
- *A diferencia de los conjuntos, las selecciones temporales no pueden guardarse en disco.*
- *Los conjuntos tienen las operación estándar Intersección, Unión y Diferencia; las selecciones temporales no pueden combinarse con otras selecciones temporales.*

Las similitudes entre las selecciones temporales y los conjuntos son:

- *Como un conjunto, una selección temporal existe en memoria.*
- *Una selección temporal y un conjunto almacenan referencias a un registro. Si los registros son modificados o borrados, la selección temporal o el conjunto se pueden volver inválidos.*
- *Como un conjunto, una selección temporal "recuerda" el registro actual del momento en que fue creada.*

CLEAR NAMED SELECTION

CLEAR NAMED SELECTION (nombre)

Parámetro	Tipo		Descripción
nombre	Cadena	➡	Nombre de la selección temporal a borrar

Descripción

CLEAR NAMED SELECTION *borra temp de la memoria y libera la memoria utilizada por temp.****CLEAR NAMED SELECTION*** *no afecta las tablas, selecciones, o registros. Como las selecciones temporales utilizan memoria, es recomendable borrar las selecciones temporales cuando ya no se necesiten.*

Si temp fue creado por el comando ***CUT NAMED SELECTION*** *y luego manipulado utilizando el comando* ***USE NAMED SELECTION****, temp no existe más en memoria. En este caso, no es necesario utilizar el comando* ***CLEAR NAMED SELECTION****.*

COPY NAMED SELECTION

COPY NAMED SELECTION ({tabla ;} nombre)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual copiar selección, o Tabla por defecto, si se omite
nombre	Cadena	➡	Nombre de la selección temporal a crear

Descripción

COPY NAMED SELECTION *copia la selección actual de tabla en una selección temporal temp. La tabla por defecto del proceso se utiliza si el parámetro opcional tabla no se especifica. El parámetro temp contiene una copia de la selección. La selección actual y el registro actual de tabla para el proceso no cambian.*

Una selección temporal no contiene en realidad registros, sino una lista ordenada de referencias a registros. Cada referencia a un registro ocupa 4 bytes en memoria. Esto significa que cuando una selección se copia utilizando el comando ***COPY NAMED SELECTION****, la cantidad de memoria necesaria es de 4 bytes multiplicado por el número de registros en la selección. Como las selecciones temporales residen en memoria, usted debe tener suficiente memoria para la selección temporal así como la selección actual de la tabla en el proceso.*

Utilice el comando ***CLEAR NAMED SELECTION*** *para liberar la memoria utilizada por temp.*

Ejemplo

El siguiente ejemplo permite verificar si hay otras facturas vencidas en la tabla [Personas]. La selección se ordena y luego se guarda. Nosotros buscamos todos los registros donde las facturas están vencidas. Luego reutilizamos la selección y borramos la selección temporal en memoria. Borrar la selección temporal en memoria es opcional, en caso que el diseñador de la base quiera conservar la selección ordenada para uso futuro:

```
ALL RECORDS([Personas])
  `Permitir al usuario ordenar la selección
ORDER BY([Personas])
  ` Guardar la selección ordenada como una selección temporal
COPY NAMED SELECTION([Personas];"OrdenUsuario")
  ` Buscar los registros donde las facturas estén vacías
QUERY([Personas];[Personas]FacturaVence=True)
  `Si se encuentran registros
If(Records in selection([Personas])>0)
  ` Informar al usuario
   ALERT("Sí, hay facturas vencidas en la tabla.")
End if
  ` Reutilizar la selección temporal ordenada
USE NAMED SELECTION("OrdenUsuario")
  ` Borrar la selección de la memoria
CLEAR NAMED SELECTION("OrdenUsuario")
```

CUT NAMED SELECTION

CUT NAMED SELECTION ({tabla ;} nombre)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual cortar selección, o Tabla por defecto, si se omite
nombre	Cadena	➡	Nombre de la selección temporal a crear

Descripción

CUT NAMED SELECTION crea una selección temporal *temp* y coloca allí la selección actual de *tabla*. A diferencia de **COPY NAMED SELECTION**, este comando no copia la selección actual de *tabla*, si no que la desplaza.

Después de ejecutar el comando, la selección actual de *tabla* en el proceso actual queda vacía. Por lo tanto, **CUT NAMED SELECTION** no debe ser utilizado cuando un registro está siendo modificado.

CUT NAMED SELECTION es más eficiente en términos de utilización de memoria, que **COPY NAMED SELECTION**. **COPY NAMED SELECTION**, utiliza 4 bytes de memoria por cada registro de la selección. **CUT NAMED SELECTION**, sólo se desplaza la referencia a la lista.

Ejemplo

El método siguiente vacía la selección actual de una tabla *[Clientes]*:

```
CUT NAMED SELECTION([Clientes];"ABorrar")
CLEAR NAMED SELECTION("ABorrar")
```

USE NAMED SELECTION

USE NAMED SELECTION (nombre)

Parámetro	*Tipo*		*Descripción*
nombre	*Cadena*	➡	*Nombre de la selección temporal a utilizar*

Descripción

USE NAMED SELECTION *utiliza el nombre de la selección temporal como selección actual para la tabla a la cual pertenece. Cuando crea una selección temporal, el registro actual es "recordado" por la selección temporal.* ***USE NAMED SELECTION*** *recupera la posición de este registro y lo vuelve el nuevo registro actual; este comando carga el registro actual. Si el registro actual fue modificado después de la creación de temp, el registro debe guardarse antes de ejecutar* ***USE NAMED SELECTION,*** *para evitar perder la información modificada.*

- *Si* ***COPY NAMED SELECTION*** *se utilizó para crear temp, la selección temporal temp se copia a la selección actual de la tabla a la cual pertenece temp. La selección temporal temp existe en memoria hasta que se borra. Utilice el comando* ***CLEAR NAMED SELECTION*** *para borrar la selección temporal y liberar la memoria utilizada por temp.*
- *Si* ***CUT NAMED SELECTION*** *fue utilizado para crear temp, temp se utiliza como selección actual de la tabla y temp no existe más en memoria.*

Recuerde que una selección temporal es una representación de una selección de registros en el momento que se crea la selección temporal. Si los registros que la selección temporal representa son modificados, la selección temporal podría ser obsoleta. Por lo tanto, una selección temporal representa un grupo de registros que no cambian con frecuencia. Diferentes cosas pueden invalidar una selección temporal: la modificación o eliminación de un registro de la selección temporal, o la modificación de los criterios de creación de la selección temporal.

Servicios Web (Cliente)

- *Comandos del tema Web Services (Client)*
- *WEB SERVICE AUTHENTICATE*
- *WEB SERVICE CALL*
- *WEB SERVICE Get info*
- *WEB SERVICE GET RESULT*
- *WEB SERVICE SET OPTION*
- *WEB SERVICE SET PARAMETER*

Comandos del tema Web Services (Client)

A partir de la versión 2003, 4D soporta "Servicios web", lo que significa que el programa permite publicar (parte servidor) y/o utilizar (parte cliente) los servicios web directamente desde su base de datos.

Un servicio web es un conjunto de funciones publicadas en una red. Estas funciones pueden ser llamadas y utilizadas por toda aplicación compatible con servicios web y conectada a una red. Los servicios web puede efectuar todo tipo de tareas, como la supervisión de entrega de paquetes de una transportador, el comercio electrónico, el monitoreo de mercado de valores, etc.

Para mayor información sobre el concepto y la operación de los servicios web, consulte el Manual de Diseño.

La subscripción a los servicios web con 4D se efectúa simplemente con la ayuda del Asistente de servicios web. En la mayoría de los casos, este Asistente será suficiente para permitirle utilizar los servicios web. Sin embargo, si quiere personalizar algunos mecanismo, debe utilizar los comandos SOAP client de 4D.

*Esta sección describe los comandos utilizados para suscripción a servicios web externos (**parte client**). Para mayor información sobre los comandos utilizados para la publicación de servicios web (parte servidor), consulte el tema .*

***Nota**: por convención, los términos "SOAP" y "Servicios web" han sido utilizados respectivamente en los nombres de los comandos (y las constantes) del lado del servidor y del lado del cliente. Estos dos conceptos hacen referencia a la misma tecnología.*

WEB SERVICE AUTHENTICATE

WEB SERVICE AUTHENTICATE (nombre ; contraseña {; metAutenticacion} {; *})

Parámetro	Tipo		Descripción
nombre	Cadena	➡	Nombre del usuario
contraseña	Cadena	➡	Contraseña del usuario
metAutenticacion	Entero largo	➡	Método de Autenticación 0 u omitido = no especificado, 1 = BASIC, 2 = DIGEST
*	Operador	➡	Si se pasa: autenticación por proxy

Descripción

*El comando **WEB SERVICE AUTHENTICATE** permite utilizar los servicios web que necesitan de la autenticación de la aplicación cliente. Los métodos BASIC y DIGEST son soportados.*

***Nota:** para más información sobre los métodos de autenticación BASIC y DIGEST, por favor revise la sección **Seguridad de las conexiones**.*

*En los parámetros nombre y contraseña, pase la información de identificación requerida (nombre de usuario y contraseña). Esta información será codificada y añadida a la petición HTTP enviada al servicio web utilizando el comando **WEB SERVICE CALL**. Es entonces necesario llamar al comando **WEB SERVICE AUTHENTICATE** antes de llamar al comando **WEB SERVICE CALL**.*

*La información de autenticación se reinicializa en cero después de cada petición. Por lo tanto, debe utilizar el comando **WEB SERVICE AUTHENTICATE** antes de llamar al comando **WEB SERVICE CALL**.*

*El parámetro opcional metAutenticacion permite indicar el método de autenticación a utilizar para la próxima llamada del comando **WEB SERVICE CALL**. Puede pasar uno de los siguientes valores:*

- *2 = utilizar el método de autenticación DIGEST*
- *1 = utilizar el método de autenticación BASIC*
- *0 (ó se omite el parámetro) = utilizar el método apropiado. En este caso, 4D envía una petición adicional para negociar el método de autenticación.*

*Por defecto, la información de autenticación se reinicia después de cada petición. Por lo tanto, debe utilizar el comando **WEB SERVICE AUTHENTICATE** antes de cada comando **WEB SERVICE CALL**. Sin embargo es posible conservar temporalmente esta información utilizando una opción del comando **WEB SERVICE SET OPTION**. En este caso, no es necesario ejecutar el comando **WEB SERVICE AUTHENTICATE** antes de cada **WEB SERVICE CALL**.*

*Si la autenticación falla, el servidor SOAP devuelve un error que puede identificar utilizando el comando **WEB SERVICE Get info**.*

Ejemplo

Autenticación con un servicio web ubicado detrás de un proxy:

```
// Autenticación al servicio web en modo DIGEST
WEB SERVICE AUTHENTICATE("SoapUser";"123";2)
// Autenticación al proxy en modo por defecto
WEB SERVICE AUTHENTICATE("ProxyUser";"456";*)
WEB SERVICE CALL(...)
```

WEB SERVICE CALL

WEB SERVICE CALL (urlAcceso ; soapAccion ; nomMetodo ; nomEspacio {; tipoCompuesto {; *}})

Parámetro	Tipo		Descripción
urlAcceso	Cadena	➡	URL de acceso al servicio Web
soapAccion	Cadena	➡	Contenido del campo SOAPAction
nomMetodo	Cadena	➡	Nombre del método
nomEspacio	Cadena	➡	Espacio del nombre (Namespace)
tipoCompuesto	Entero largo	➡	Configuración de tipos compuestos (tipos simples si se omite)
*	Operador	➡	No cerrar la conexión

Descripción

*El comando **WEB SERVICE CALL** se utiliza para llamar un servicio web enviando una petición HTTP. Esta petición contiene el mensaje SOAP creado previamente utilizando el comando **WEB SERVICE SET PARAMETER**.*

*Toda llamada posterior al comando **WEB SERVICE SET PARAMETER** provocará la creación de una nueva petición. La ejecución de un comando **WEB SERVICE CALL** también borra todo resultado del servicio web llamado anteriormente y lo reemplaza con los nuevos resultados.*

En urlAcceso, pase el URL completo que permite acceder al servicio web (no confunda este URL con el del archivo WSDL, que describe el servicio web).

- ***Acceso en modo seguro (SSL)***

Si quiere utilizar un servicio web en modo seguro utilizando SSL, pase https:// delante del URL en lugar de http://. Esta configuración activa automáticamente la conexión en modo seguro.

*Note que este comando puede utilizar un certificado servidor (ver el comando **HTTP SET CERTIFICATES FOLDER**). Si este certificado no es valido (vencido o revocado), la variable sistema OK toma el valor y se devuelve el error 901 "Certificado servidor invalido". Puede interceptar este error utilizando un método de gestión de errores instalado por el comando **ON ERR CALL**.*

*En soapAccion, pase el contenido del campo SOAPAction de la petición. Este campo contiene por lo general el valor "**ServiceName#MethodName**".*

En nomMetodo, pase el nombre del método remoto (que pertenece al servicio web) que quiere ejecutar.

En espacioNombre, pase el espacio del nombre XML (namespace) utilizado para la petición SOAP. Para mayor información sobre los nombres de espacios XML, consulte el Manual de Diseño.

*El parámetro opcional tipoCompuesto especifica la configuración de los parámetros web Service enviados o recibidos (definidos utilizando los comandos **WEB SERVICE SET PARAMETER** y **WEB SERVICE GET RESULT**.*
El valor del parámetro tipoCompuesto depende del modo de publicación del servicio web (DOC o RPC, ver el Manual de Diseño) y sus propios parámetros.

*En tipoCompuesto, debe pasar una de las siguientes constantes, ubicadas en el tema **Servicios Web (Cliente)**:*

Constante	Tipo	Valor
Web Service dynamic	Entero largo	0
Web Service manual	Entero largo	3
Web Service manual in	Entero largo	1
Web Service manual out	Entero largo	2

Cada constante corresponde a una "configuración". Una configuración representa una combinación entre el modo de publicación (RPC/DOC) y los tipos de parámetros (entrada/salida, simple o compuesto) implementado.

***Nota:** recuerde que la característica "entrada" o "salida" de los parámetros se evalúa desde el punto de vista del método proxy/servicio web:*

- *un parámetro "entrada" es un valor pasado al método proxy y por lo tanto al servicio web,*
- *un parámetro "salida" es devuelto por el servicio web y por lo tanto por el método proxy (generalmente vía $0).*

La siguiente tabla muestra todas las configuraciones posibles como también las constantes correspondientes:

	Parámetros entrada	
Parámetros entrada	**Simples**	**Compuestos**
Simples	Web Service Dynamic (Modo RPC)	Web Service Manual In (Modo RPC)
Compuestos	Web Service Manual Out (Modo RPC)	Web Service Manual (Modo RPC o Modo DOC)

Las cinco configuraciones descritas a continuación pueden implementarse. En todos los caso, 4D administrará el formato de la petición SOAP a enviar al servicio web como también su sobre. Es su decisión darle formato a los contenidos de esta petición de acuerdo a la configuración utilizada.

***Nota:** a pesar del hecho de que los tipos XML compuestos, los arrays de datos son tratados por 4D como tipos simples.*

Modo RPC, entrada y salida simples

Esta configuración es la más fácil de utilizar. En este caso, el parámetro tipoCompuesto contiene la constante Web Service Dynamic o se omite.

Los parámetros enviados y las respuestas recibidas pueden ser manipulados directamente, sin procesamiento previo.

*Consulte el ejemplo del comando **WEB SERVICE GET RESULT**.*

Modo RPC, entrada compuesta y salida simple

En este caso, el parámetro tipoComplejo contiene la constante Web Service Manual In. Con esta configuración, debe pasar "manualmente" al servicio Web cada elemento XML fuente bajo la forma de un BLOB, con la ayuda del comando ***WEB SERVICE SET PARAMETER****.*

Depende de usted formatear el BLOB inicial como un elemento XML válido. Este BLOB debe contener como primer elemento el primer elemento "hijo" del elemento <Body> de la petición final.

Ejemplo

```
C_BLOB($1)
C_BOOLEAN($0)

WEB SERVICE SET PARAMETER("MyXMLBlob";$1)
WEB SERVICE CALL("http://my.domain.com/my_service";"MiAccionSoap";"ElMetodo";"http://mi.nombrespacio.com/";Web Service manual in)
WEB SERVICE GET RESULT($0;"MiVarSalida";*)
```

Modo RPC, entrada simple y salida compuestas

En este caso, el parámetro tipoCompuesto contiene la constante Web Service Manual Out. Cada parámetro de salida será devuelto por el servicio Web bajo la forma del elemento XML almacenado en un BLOB. Recupera este parámetro utilizando el comando ***WEB SERVICE GET RESULT****. Luego puede analizar el contenido del BLOB recibido utilizando los comandos XML de 4D.*

Ejemplo

```
C_BLOB($0)
C_BOOLEAN($1)

WEB SERVICE SET PARAMETER("MiVarEntrada";$1)
WEB SERVICE CALL("http://mi.dominio.com/mi_servicio";"MiAccionSoap";"ElMetodo";"http://mi.nombrespacio.com/";Web Service manual out)
WEB SERVICE GET RESULT($0;"MiXMLSalida";*)
```

Modo RPC, entrada y salida compuestas

En este caso, el parámetro tipoCompuesto contiene la constantes Web Service Manual. Cada parámetro de entrada y de salida debe ser almacenado en la forma de los elementos XML en los BLOBs, como se describió en las dos configuraciones anteriores.

Ejemplo

```
C_BLOB($0)
C_BLOB($1)

WEB SERVICE SET PARAMETER("MiBlobXMLEntrada";$1)
WEB SERVICE CALL("http://mi.dominio.com/mi_servicio";"MiAccionSoap";"ElMetodo";"http://mi.nombrespacio.com/";Web Service manual)
WEB SERVICE GET RESULT($0;"MiXMLSalida";*)
```

Modo DOC

Un método proxy de llamada de un servicio web DOC es similar a un método proxy de llamada de un servicio web RPC utilizando los parámetros de entrada y de salida compuestos.

La única diferencia entre estas dos configuraciones es el nivel del contenido XML de los parámetros BLOB pasados y recibidos. Desde el punto de vista de 4D, la construcción y el envío de la petición SOAP son idénticos.

Ejemplo

```
C_BLOB($0)
C_BLOB($1)

WEB SERVICE SET PARAMETER("MiXMLEntrada";$1)
WEB SERVICE CALL("http://mi.dominio.com/mi_servicio";"MiAccionSoap";"ElMetodo";"http://mi.nombrespacio.com/";Web Service manual)
WEB SERVICE GET RESULT($0;"MiXMLSalida";*)
```

Nota: *en el caso de los servicios web DOC, el valor de las cadenas ("MiXMLEntrada" y "MiXMLSalida") pasadas como parámetros no es de importancia; incluso es posible pasar cadenas vacías"". De hecho, estos valores no se utilizan en la petición SOAP contenida en el documento XML. Es obligatorio pasar estos parámetros.*

Para utilizar un servicio web publicado en modo DOC (o en modo RPC con tipos compuestos), es recomendable proceder de esta forma:

- *Generar el método proxy utilizando el Asistente Client Web Services.*
  El método proxy se llamará de esta forma: $XMLresultadoBlob:=$DOCproxy_Metodo($XMLparamBlob)
- *Familiariarisece con los contenidos de las peticiones SOAP a enviar al servicio web utilizando una herramienta de prueba en línea (por ejemplo, http://soapclient.com/soaptest.html). Este tipo de herramienta se utiliza para generar los formularios de prueba HTML, a partir del WSDL del servicio web.*
- *Copie el contenido XML generado a partir del primer hijo de <body>.*
- *Escriba el método permitiendo ubicar los valores reales de los parámetros en el código XML; este código debe estar ubicado en el BLOB $XMLparamBlob.*
- *Para analizar la respuesta, puede igualmente utilizar una herramienta de prueba en línea, o utilizar el WSDL que especifica los elementos devueltos.*

*El parámetro * puede utilizarse para optimizar llamadas. Cuando se pasa, el comando no cierra la conexión utilizada por el proceso al final de su ejecución. En este caso, la próxima llamada a* ***WEB SERVICE CALL*** *reutilizará la misma conexión si se pasa el parámetro *, etc. Para cerrar la conexión, simplemente ejecute el comando* ***WEB SERVICE CALL*** *sin el parámetro *. Este mecanismo puede utilizarse para acelerar sensiblemente los procesos en caso de llamadas sucesivas a varios servicios web en el mismo servidor, en particular en una configuración WAN (vía Internet, por ejemplo). Note que este mecanismo depende del parámetro "keep-alive" del servidor web. Este parámetro por lo general define un número máximo de peticiones vía la misma conexión, e incluso puede negar peticiones. Si las peticiones* ***WEB SERVICE CALL*** *encadenadas en la misma conexión alcanzan este número máximo, o si las conexiones keep-alive no son permitidas, 4D creará una nueva conexión para cada petición.*

Variables y conjuntos del sistema

Si la petición se enruta correctamente y el servicio web la acepta, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0 y se devuelve un error.

WEB SERVICE Get info

WEB SERVICE Get info (tipoInfo) -> Resultado

Parámetro	Tipo		Descripción
tipoInfo	Entero largo	→	Información a recuperar
Resultado	Cadena	↩	Información sobre el último error SOAP

Descripción

*El comando **WEB SERVICE Get info** devuelve la información sobre todo error encontrado durante la ejecución de la última petición SOAP enviada a un servicio web remoto. Generalmente, este comando debe llamarse dentro de un método de gestión de errores instalado por el comando **ON ERR CALL**. El parámetro tipoInfo le permite indicar el tipo de información que quiere obtener. Debe pasar una de las constantes listadas a continuación, ubicada en el tema **Servicios Web (Cliente)**:*

Constante	Tipo	Valor	Comentario
Web Service detailed message	Entero largo	1	Mensaje detallado que describe el error. El tipo de mensaje difiere según el tipo de error principal. - Si el error principal = 9910 (Error Soap): se devuelve la causa del error SOAP (ej.: "el método remoto no existe"). - Si el error principal = 9911 (Error de analizador xml): se devuelve la ubicación del error en el documento XML. - Si el error principal = 9912 (Error HTTP): - Si el error HTTP se ubica en el intervalo [300-400] (problemas relacionados con la ubicación del documento solicitado), se devuelve la nueva ubicación del URL solicitado. - Para todo otro código de error HTTP, se devuelve el <body>. - Si el error principal = 9913 (Error de red): se devuelve la causa del error de red (ej.: "ServerAddress: error DNS") - Si el error principal = 9914 (Error interno): se devuelve la causa del error interno
Web Service error code	Entero largo	0	Código del error principal (definido por 4D). Este código también es devuelto en la variable sistema Error. Lista de códigos que pueden ser devueltos: 9910: Error Soap (ver también Web Service Fault Actor) 9911: Error de analizador xml 9912: Error HTTP (ver también Web Service HTTP Error code) 9913: Error red 9914: Error interno.
Web Service fault actor	Entero largo	3	Causa del error (devuelto por el protocolo SOAP, a utilizar en caso de error principal 9910). - Version Mismatch - Must Understand (un parámetro definido como obligatorio no puede ser interpretado por el servidor) - Sender Fault - Receiver Fault - Encoding Unknown
Web Service HTTP error code	Entero largo	2	Código del error HTTP (a utilizar en caso de error principal 9912).

Se devuelve una cadena vacía cuando no hay información disponible, en particular si la última petición SOAP no generó errores.

WEB SERVICE GET RESULT

WEB SERVICE GET RESULT (valorDevuelto {; nombreDevuelto {; *}})

Parámetro	Tipo		Descripción
valorDevuelto	Variable	⇐	Valor devuelto por el servicio web
nombreDevuelto	Cadena	⇒	Nombre del parámetro a recuperar
*		⇒	Liberar memoria

Descripción

*El comando **WEB SERVICE GET RESULT** permite recuperar un valor enviado por el servicio web como resultado del proceso realizado.*

***Nota:** este comando debe utilizarse únicamente después del comando **WEB SERVICE CALL**.*

El parámetro valorDevuelto recibe el valor reenviado por el servicio web. Pase en este parámetro una variable 4D. Esta variable es generalmente $0, que corresponde al valor devuelto por el método proxy. Sin embargo, es posible utilizar variables intermediarias (debe utilizar las variables de proceso únicamente).

***Nota:** cada variable 4D o array utilizado debe ser declarado previamente utilizando los comandos de los temas "Compilador" y "Arrays".*

El parámetro opcional nombreDevuelto se utiliza para especificar el nombre del parámetro a recuperar. Sin embargo, como la mayoría de los servicios web devuelven un solo valor, por lo general este parámetro no es necesario.

*El parámetro opcional *, indica al programa que libere la memoria dedicada al procesamiento de la petición. Debe pasar este parámetro después de recuperar el último valor enviado por el servicio web.*

Ejemplo

Imagine un servicio web que devuelve la hora actual en cualquier ciudad del mundo. Los parámetros recibidos por el servicio web son el nombre de la ciudad y el código del país. El servicio web devuelve la correspondiente. El método proxy de llamada puede ser de la siguiente forma:

```
C_TEXT($1)
C_TEXT($2)
C_TIME($0)</p>
WEB SERVICE SET PARAMETER("ciudad";$1)
WEB SERVICE SET PARAMETER("codigo_pais";$2)

WEB SERVICE
CALL("http://www.ciudadesdelmundo.com/WS";"WSTime#City_time";"City_time";"http://www.ciudadesdelmundo.com/namespace/default")

If(OK=1)
   WEB SERVICE GET RESULT($0;"devolver";*)
End if
```

WEB SERVICE SET OPTION

WEB SERVICE SET OPTION (opción ; valor)

Parámetro	Tipo		Descripción
opción	Entero largo	➡	Código de la opción a definir
valor	Entero largo, Texto	➡	Valor de la opción

Nota preliminar

Este comando está diseñado para los usuarios de servicios web. Su uso es opcional.

Descripción

*El comando **WEB SERVICE SET OPTION** permite definir diferentes opciones que se utilizarán durante la próxima petición SOAP provocada por el comando **WEB SERVICE CALL**.*

Puede llamar este comando tantas veces como opciones a definir.

*En el parámetro opcion, pase el número de la opción a definir y en el parámetro valor, pase el nuevo valor de la opción. Para estos parámetros, puede utilizar una de las siguientes constantes predefinidas del tema **Servicios Web (Cliente)**:*

Constante	Tipo	Valor	Comentario
Web Service display auth dialog	Entero largo	4	valor = 0 (no mostrar la caja de diálogo) ó 1 (mostrar caja de diálogo) Esta opción administra la visualización de la caja de diálogo de actualización durante la ejecución del comando **CALL WEB SERVICE**. Por defecto, este comando nunca muestra la caja de diálogo; por lo general, para hacerlo debe utilizar el comando **AUTHENTICATE WEB SERVICE**. Sin embargo, si quiere que aparezca la caja de diálogo de autenticación para que el usuario introduzca sus identificadores, deberá utilizar esta opción: pase 1 en valor para mostrar la caja de diálogo, de lo contrario pase 0. La caja de diálogo sólo aparece si el servicio web necesita autenticación.
Web Service HTTP compression	Entero largo	6	valor = Web Service Compression Esta opción permite activar un mecanismo interno de compresión de las peticiones SOAP con el fin de acelerar los intercambios entre aplicaciones 4D. Cuando ejecuta la instrucción **WEB SERVICE SET OPTION**(Web Service HTTP Compression; Web Service Compression) en el cliente 4D del servicio web, los datos de la próxima petición SOAP enviados por el cliente serán comprimidos utilizando un mecanismo estándar HTTP ("gzip" o "deflate" en función del contenido de la petición) antes de su envío al servidor SOAP 4D. El servidor descomprimirá y analizará la petición, luego responderá automáticamente utilizando el mismo mecanismo. Sólo se afecta la petición que sigue la llamada al comando **WEB SERVICE SET OPTION**. Por lo tanto debe llamar este comando cada vez que quiera utilizar la compresión. Por defecto, 4D no comprime las peticiones HTTP de los servicios web. **Nota:** este mecanismo no puede utilizarse para las peticiones enviadas a un servidor SOAP 4D de una versión anterior a la 11.3. Para que pueda optimizar más este funcionamiento, las opciones adicionales configuran el límite y la tasa de compresión de las peticiones. Estas opciones son accesibles vía el comando **SET DATABASE PARAMETER**.
Web Service HTTP timeout	Entero largo	1	valor = "timeout" de la parte cliente expresado en segundos. El timeout de la parte clientes es el periodo de espera del cliente servicio web en caso de que no haya respuesta del servidor. Después de este período, el cliente cierra la sesión y se pierde la petición. Por defecto, este timeout es de 180 segundos. Puede modificarse por razones específicas (estado de la red, especificaciones del servicio web, etc.).
Web Service reset auth settings	Entero largo	5	valor = 0 (no borrar la información) ó 1 (borra la información) Esta opción le permite indicar a 4D memorizar la información de autenticación del usuario (nombre de usuario, contraseña, método, etc.), para reutilizarla posteriormente. Por defecto, esta información se borra después de cada ejecución del comando **CALL WEB SERVICE**. Pase 0 en valor para guardar la información y 1 para borrarla. Note que cuando pasa 0, la información se conserva durante la sesión pero no se almacena.
Web Service SOAP header	Entero largo	2	valor = referencia del elemento XML raíz a insertar como encabezado de la petición SOAP. Esta opción permite insertar un encabezado en la petición SOAP generada utilizando el comando **CALL WEB SERVICE**. Por defecto, las peticiones SOAP no contienen un encabezado específico. Sin embargo, algunos servicios web requieren un encabezado, por ejemplo para la gestión de los parámetros de identificación.
Web Service SOAP version	Entero largo	3	valor = Web Service SOAP_1_1 o Web Service SOAP_1_2 Esta opción permite precisar la versión del protocolo SOAP utilizado en la petición. Pase en valor la constante Web Service SOAP_1_1 para indicar la versión 1.1 y la constante Web Service SOAP_1_2 para indicar la versión 1.2.

El orden de llamada de las opciones no es importante. Si la misma opcion está definida varias veces, sólo el valor de la primera llamada se tiene en cuenta.

Ejemplo 1

Inserción de un encabezado personalizado en la petición SOAP:

```
  ` Creación de una referencia XML
C_TEXT(vRootRef;vElemRef)
vRootRef:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vElemRef:=DOM Create XML element(vRootRef;vxPath)
  `Modificación del encabezado SOAP con la referencia
WEB SERVICE SET OPTION(Web Service SOAP header;vElemRef)
```

Ejemplo 2

Utilización de la versión 1.2 del protocolo SOAP:

```
WEB SERVICE SET OPTION(Web Service SOAP version;Web Service SOAP_1_2)
```

WEB SERVICE SET PARAMETER

WEB SERVICE SET PARAMETER (nombre ; valor {; tipoSOAP})

Parámetro	Tipo		Descripción
nombre	Cadena	➡	Nombre del parámetro a incluir en la petición SOAP
valor	Variable	➡	Variable 4D que contiene el valor del parámetro
tipoSOAP	Cadena	➡	Tipo SOAP del parámetro

Descripción

*El comando **WEB SERVICE SET PARAMETER** permite la definición de un parámetro utilizado por una petición SOAP cliente. Llame este comando por cada parámetro en la petición (el número de veces que se llame el comando depende del número de parámetros).*

Pase en nombre el nombre del parámetro tal como aparece en la petición SOAP.

En valor, pase la variable 4D que contiene el valor del parámetro. En el caso de los métodos proxy, esta variable es generalmente $1, $2, $3, etc., correspondiente a un parámetro 4D pasado al método proxy durante su llamada. Sin embargo, es posible utilizar variables intermediarias.

***Nota:** cada variable o array 4D utilizado debe declararse previamente utilizando los comandos de los temas **Compilador** y **Arrays**.*

Por defecto, 4D determina automáticamente el tipo SOAP más apropiado para el parámetro nombre de acuerdo al contenido de valor. La indicación del tipo está incluida en la petición.

Sin embargo, podría "forzar" la definición del tipo SOAP de un parámetro. En este caso, puede pasar el parámetro opcional tipoSOAP utilizando una de las siguientes cadenas de caracteres (tipos de datos primarios):

tipoSOAP	Descripción
string	Cadena
int	Entero largo
boolean	Booleano
float	Real 32 bits
decimal	Real con décimal
double	Real 64 bits
duration	Duración en años, meses, días, horas, minutos, segundos, por ejemplo: P1Y2M3DT10H30M
datetime	Fecha y hora en formato ISO8601, por ejemplo 2003-05-31T13:20:00
time	Hora, por ejemplo 13:20:00
date	Fecha, por ejemplo 2003-05-31
gyearmonth	Año y mes (calendario gregoriano), por ejemplo 2003-05
gyear	Año (calendario gregoriano), por ejemplo 2003
gmonthday	Mes y día (calendario gregoriano), por ejemplo --05-31
gday	Día (calendario gregoriano), por ejemplo ---31
gmonth	Mes (calendario gregoriano), por ejemplo --10--
hexbinary	Valor expresado en hexadecimal
base64binary	BLOB
anyuri	Uniform Resource Identifier (URI), por ejemplo http://www.empresa.com/page
qname	Nombre XML calificado (espacio de nombre y parte local)
notation	Atributo notación

***Nota:** para mayor información sobre tipos de datos XML, consulte el URL http://www.w3.org/TR/xmlschema-2/*

Ejemplo

Este ejemplo define dos parámetros:

```
C_TEXT($1)
C_TEXT($2)
WEB SERVICE SET PARAMETER("ciudad";$1)
WEB SERVICE SET PARAMETER("país";$2)
```

Servicios Web (Servidor)

Web Services (Servidor)
Is SOAP request
SOAP DECLARATION
SOAP get info
SOAP SEND FAULT

Web Services (Servidor)

La publicación de servicios web con 4D se efectúa simplemente utilizando las opciones en las propiedades de los métodos. En la mayoría de los casos, esta operación será suficiente para permitirle publicar servicios web. Sin embargo, si quiere personalizar ciertos mecanismos, utilizar los arrays de datos, etc., debe utilizar los comandos SOAP servidor de 4D.

*Esta sección describe los comandos utilizados para la publicación de servicios web en 4D (**parte servidor**). Para mayor información sobre servicios web o para una descripción de los comandos utilizados para la suscripción a servicios web (parte cliente), consulte la sección **Servicios Web (Cliente)**.*

***Nota**: por convención, los términos "SOAP" y "Web Service" han sido utilizados respectivamente en los nombres de los comandos (y de las constantes) del lado del servidor y del cliente por convención únicamente. Estos dos conceptos hacen referencia a la misma tecnología.*

Is SOAP request

Is SOAP request -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*True si la petición es SOAP; de lo contrario, False*

Descripción

*El comando **Is SOAP request** devuelve **True** si el código en ejecución hace parte de una petición SOAP.*
*Este comando puede utilizarse por razones de seguridad en el **Método de base On Web Authentication** para determinar la naturaleza de las peticiones recibidas.*

SOAP DECLARATION

SOAP DECLARATION (variable ; tipo ; entrada_salida {; alias})

Parámetro	Tipo		Descripción
variable	Variable	➡	Variable que referencia un argumento SOAP entrante o saliente
tipo	Entero largo	➡	Tipo 4D al cual apunta el argumento
entrada_salida	Entero largo	➡	1 = Entrada SOAP, 2 = Salida SOAP
alias	Cadena	➡	Nombre publicado para este argumento durante los intercambios SOAP

Descripción

*El comando **SOAP DECLARATION** permite declarar explícitamente el tipo de los parámetros utilizados en un método 4D publicado como servicio web.*

Cuando se publica un método como servicio web, los parámetros estándar $0, $1... $n se utilizan para describir los parámetros del servicio web al mundo exterior (más particularmente en el archivo WSDL). El protocolo SOAP necesita que los parámetros sean nombrados explícitamente; 4D utiliza por defecto los nombres "FourD_arg0, FourD_arg1 ... FourD_argn".

Este funcionamiento por defecto puede ser problemático por las siguientes razones:

- *No es posible declarar $0 o $1, $2, etc. como un array. Por lo tanto, es necesario utilizar punteros; sin embargo, en este caso, es necesario conocer el tipo de los valores para la generación del archivo WSDL.*
- *Luego, puede ser útil o necesario personalizar los nombres de los parámetros (entrantes y salientes).*
- *Si lo desea, puede utilizar parámetros como las estructuras XML y referencias DOM.*
- *Igualmente, es necesario devolver los valores de un tamaño superior a 32 KB (limite de los argumentos de tipo Texto).*
- *Por último, este funcionamiento hace imposible devolver más de un valor por llamada RPC (en $0).*

*El comando **SOAP DECLARATION** le permite liberar estos límites. Puede ejecutar este comando para cada parámetro entrante y saliente y asignarle un nombre y un tipo.*

***Nota:** incluso si se utiliza el comando **SOAP DECLARATION**, siempre es necesario declarar las variables y los arrays 4D ven el método Compiler_Web utilizando los comandos del tema "".*

En variable, pase la variable 4D a referenciar cuando llame al servicio web.

***Advertencia:** puede referenciar únicamente las variables proceso o los argumentos de los métodos 4D ($0 a $n). Las variables locales e interproceso no pueden utilizarse.*

*Por defecto, como sólo pueden utilizarse los argumento de tipo texto, las respuestas del servidor SOAP están limitadas a 32 KB. Sin embargo, es posible devolver argumentos SOAP con un tamaño mayor a 32 KB, utilizando BLOBs. Para exceder este límite, simplemente necesita declarar los argumentos como BLOBs antes de llamar al comando **SOAP DECLARATION** (ver ejemplo 4).*

***Nota:** del lado del cliente, si suscribe este tipo de servicio web con 4D, el asistente de servicios web generará naturalmente una variable de tipo Texto en el método proxy. Para poder utilizarla, sólo necesita digitar nuevamente esta variable como un BLOB en el método proxy.*

En tipo, pase el tipo 4D correspondiente. Pueden ser utilizados la mayoría de los tipos de variables y arrays 4D. Puede utilizar las siguientes constantes predefinidas, ubicadas en el tema "":

Constante	Tipo	Valor
Is real	Entero largo	1
Is text	Entero largo	2
Is date	Entero largo	4
Is Boolean	Entero largo	6
Is integer	Entero largo	8
Is longint	Entero largo	9
Is time	Entero largo	11
Real array	Entero largo	14
Integer array	Entero largo	15
LongInt array	Entero largo	16
Date array	Entero largo	17
Text array	Entero largo	18
String array	Entero largo	21
Boolean array	Entero largo	22
Is string var	Entero largo	24
Is BLOB	Entero largo	30

Constante	Tipo	Valor
Is XML	Entero largo	36
Is DOM reference	Entero largo	37

*En entrada_salida, pase un valor indicando si el parámetro procesado es "entrante" (es decir corresponde a un valor recibido por el método) o "saliente" (por ejemplo corresponde a un valor devuelto por el método). Puede utilizar las siguientes constantes predefinidas, ubicadas en el tema "**Web Services (Servidor)**":*

Constante	Tipo	Valor
SOAP input	Entero largo	1
SOAP output	Entero largo	2

Utilización de tipos XML

Puede declarar las variables de tipo "estructura XML" y "DOM reference", de entrada y salida, vía las constantes Is XML y Is DOM reference. Cuando los parámetros de este tipo son definidos, no se aplica ningún proceso ni codificación y los datos se trasmiten "tal cual" (ver ejemplo 5).

- ***Parámetros de salida***

  - Is XML indica que el parámetro contiene una estructura XML,
  - Is DOM reference indica que el parámetro contiene la referencia DOM de una estructura XML. En este caso, la inserción de la estructura XML en el mensaje SOAP es
  equivalente a ejecutar el comando ***DOM EXPORT TO VAR****.*

Nota: *en el caso de las referencias DOM utilizadas como parámetros de salida, se recomienda utilizar las referencias globales, creadas, por ejemplo, al inicio o al cierre de la aplicación. De hecho, una referencia DOM creada dentro del mismo servicio web no puede cerrarse con* ***DOM CLOSE XML****, de lo contrario el servicio web no devuelve nada. Las llamadas múltiples al servicio web implican por lo tanto la creación de múltiples referencias DOM no cerradas, las cuales pueden provocar una saturación de la memoria.*

- ***Parámetros de entrada***

  - Is XML indica que el parámetro debe recibir un argumento XML enviado por el cliente SOAP.
  *- Is DOM reference indica que el parámetro debe recibir la referencia DOM de una estructura XML correspondiente al argumento XML enviado por el cliente SOAP****.***

- ***Modificación del WSDL***

  Las estructuras XML serán declaradas por 4D como "anyType" (tipo indeterminado) en el WSDL. Si quiere dar un tipo preciso a una estructura XML, debe guardar el archivo WSDL y añadir manualmente el esquema de datos que quiere en la sección <types> del WSDL.

Método COMPILER_WEB

Los argumentos SOAP entrantes referenciados con la ayuda de variables 4D (y no por los argumentos de métodos 4D) deben ser declarados primero en el método de proyecto COMPILER_WEB. De hecho, el uso de variables de proceso en los métodos Web Services necesitan ser declaradas antes de la llamada al método. Se llama el método de proyecto COMPILER_WEB, si existe, para cada petición SOAP aceptada. Por defecto, el método COMPILER_WEB no existe. Usted debe crearlo específicamente.

Note que el método COMPILER_WEB también es llamado por el servidor web de 4D durante la recepción de peticiones web "convencionales" de tipo POST (ver la sección ***URLs y acciones de formularios****).*

En alias, pase el nombre del argumento que debe aparecer en WSDL y en los intercambios SOAP.

Advertencia: *este nombre debe ser único en la llamada RPC (parámetros entrantes y salientes), de lo contrario, sólo la última declaración será tenida en cuenta por 4D.*

Nota: *los nombres de los argumentos no deben comenzar por un número ni contener espacios. Además, para evitar riesgos de incompatibilidad, se recomienda no utilizar caracteres extendidos (tales como caracteres con acentos).*
Si se omite el parámetro alias, 4D utilizará, por defecto, el nombre de la variable o ***FourD_argN*** *para los argumentos de los métodos 4D (\$0 a \$n).*

Nota: *el comando* ***SOAP DECLARATION*** *debe estar incluido en el método publicado como servicio web. No es posible llamarlo desde otro método.*

Ejemplo 1

Este ejemplo especifica un nombre de parámetro:

```
` En el método COMPILER_WEB
C_LONGINT($1)

` En el método del servicio Web
` Durante la generación del archivo WSDL y las llamadas SOAP, la palabra
` zipcode se utilizará en lugar de fourD_arg1
SOAP DECLARATION($1;Is longint;SOAP input;"zipcode")
```

Ejemplo 2

Este ejemplo se utiliza para recuperar un array de códigos postales en forma de enteros largos:

```
`En el método COMPILER_WEB
ARRAY LONGINT(codes;0)

`En el método del servicio Web
SOAP DECLARATION(codes;LongInt array;SOAP input;"in_codes")
```

Ejemplo 3

Este ejemplo se utiliza para referenciar dos valores devueltos sin especificar un nombre de argumento:

```
SOAP DECLARATION(ret1;Is longint;SOAP output)
SOAP DECLARATION(ret2;Is longint;SOAP output)
```

Ejemplo 4

Este ejemplo permite al servidor 4D SOAP devolver un argumento de un tamaño mayor a 32 KB:

```
C_BLOB($0)
SOAP DECLARATION($0;Is text;SOAP output)
```

*Note el tipo **Is Text** (y no **Is BLOB**). Esto permite que el argumento sea procesado correctamente.*

Ejemplo 5

Este ejemplo illustra los resultados de los diferentes tipos de declaraciones:

```
ALL RECORDS([Contactos])

  `Construcción de una estructura XML de la selección de contactos y almacenar el XML en un BLOB
C_BLOB(ws_vx_xmlBlob)
obtenerContactosXML(->ws_vx_xmlBlob)
  `Recuperar la estructura XML en una variable de tipo texto
C_TEXT(ws_vt_xml)
ws_vt_xml:=BLOB to text(ws_vx_xmlBlob;UTF8 text without length)
  `Recuperar una referencia DOM a la estructura XML
C_TEXT(ws_vt_xmlRef)
ws_vt_xmlRef:=DOM Parse XML variable(ws_vt_xml)

  `Prueba las diferentes declaraciones
SOAP DECLARATION(ws_vx_xmlBlob;Is BLOB;SOAP output;"contactListsX")
  `El XML se convierte en Base64 por 4D

SOAP DECLARATION(ws_vt_xml;Is text;SOAP output;"contactListsText")
  `El XML se convierte en texto por 4D (< > become entities)

SOAP DECLARATION(ws_vt_xml;Is XML;SOAP output;"xmlContacts")
  `El XML se pasa a texto XML

SOAP DECLARATION(ws_vx_xmlBlob;Is XML;SOAP output;"blobContacts")
  `El XML se pasa a un blob XML

SOAP DECLARATION(ws_vt_xmlRef;Is DOM reference;SOAP output;"contactByRef")
  `El XML se pasa como una referencia
```

SOAP get info

SOAP get info (numInfo) -> Resultado

Parámetro	Tipo		Descripción
numInfo	Entero largo	→	Número de tipo de información SOAP a obtener
Resultado	Cadena	←	Información SOAP

Descripción

*El comando **SOAP get info** permite recuperar bajo la forma de cadena de caracteres diferentes tipos de información relacionada con una petición SOAP.*

*Cuando procesa una petición SOAP, puede ser útil obtener información adicional, diferente a los valores de los parámetros RPC, sobre la petición. Por ejemplo, por razones de seguridad, puede utilizar este comando en el **Método base On Web Authentication** para conocer el nombre del método Web service solicitado.*

*Pase en el parámetro numInfo el número del tipo de información SOAP que quiere conocer. Puede utilizar las siguientes constantes predefinidas, ubicadas en el tema **Servicios Web (Servidor)**:*

Constante	Tipo	Valor	Comentario
SOAP method name	Entero largo	1	Nombre del método ofrecido como servicio web a punto de ejecutarse
SOAP service name	Entero largo	2	Nombre del servicio web al que pertenece el método

***Nota:** por razones de seguridad, es posible obtener el tamaño máximo de las peticiones de servicios web enviadas a 4D. Esta configuración se lleva a cabo utilizando el comando **SET DATABASE PARAMETER** (Tema "Definición estructura").*

SOAP SEND FAULT

SOAP SEND FAULT (tipoError ; descripcion)

Parámetro	Tipo		Descripción
tipoError	Entero largo	➡	1 = Error cliente, 2 = Error servidor
descripcion	Cadena	➡	Descripción del error a enviar al cliente SOAP

Descripción

*El comando **SOAP SEND FAULT** devuelve un error a un cliente SOAP indicando el origen del error: cliente o servidor. Este comando permite indicar un error a un cliente sin tener que devolver un resultado.*

Por ejemplo, un error del lado del cliente puede ser detectado cuando publica un servicio web "Raiz_cuadrada" y un cliente envía una petición con un número negativo; puede utilizar este comando con el fin de indicar al cliente que se necesita un valor positivo.

Un error posible del lado del servidor podría ser por ejemplo, falta de memoria durante la ejecución del método.

*Pase el origen del error en tipoError. Puede utilizar las siguientes constantes predefinidas, ubicadas en el tema **Servicios Web (Servidor)**.*

Constante	Tipo	Valor
SOAP client fault	Entero largo	1
SOAP server fault	Entero largo	2

Pase en descripcion la descripción del error. Si la implementación del cliente es correcta, el error puede procesarse.

Ejemplo

Regresando al ejemplo del servicio Web "Raiz_cuadrada" de la descripción del comando, la siguiente instrucción puede utilizarse para procesar peticiones con números negativos:

```
SEND SOAP FAULT(SOAP client fault;"Valores positivos requeridos")
```

Servidor Web

- *Presentación del servidor web*
- *Configuración del servidor web y gestión de conexiones*
- *Soporte de IP v6*
- *Seguridad de las conexiones*
- *Método de base On Web Authentication*
- *Método base On Web Connection*
- *Método base On Web Close Process*
- *Gestión de las sesiones web*
- *Páginas semidinamicas*
- *URLs y acciones de formularios*
- *Asociar objetos 4D a objetos HTML*
- *Parámetros del servidor web*
- *Utilizar procesos web apropiativos*
- *Información sobre el sitio web*
- *Definir las páginas de errores HTTP personalizadas*
- *Utilizar el protocolo TLS (HTTPS)*
- *Soporte de XML y WML*
- *WEB CLOSE SESSION*
- *WEB GET BODY PART*
- *WEB Get body part count*
- *WEB Get Current Session ID*
- *WEB GET HTTP BODY*
- *WEB GET HTTP HEADER*
- *WEB GET OPTION* Updated 17.0
- *WEB Get server info* Updated 17.0
- *WEB GET SESSION EXPIRATION*
- *WEB Get session process count*
- *WEB GET STATISTICS*
- *WEB GET VARIABLES*
- *WEB Is secured connection*
- *WEB Is server running*
- *WEB SEND BLOB*
- *WEB SEND FILE*
- *WEB SEND HTTP REDIRECT*
- *WEB SEND RAW DATA*
- *WEB SEND TEXT*
- *WEB SET HOME PAGE*
- *WEB SET HTTP HEADER*
- *WEB SET OPTION* Updated 17.0
- *WEB SET ROOT FOLDER*
- *WEB START SERVER*
- *WEB STOP SERVER*
- *WEB Validate digest*
- *_o_SET CGI EXECUTABLE*
- *_o_SET WEB DISPLAY LIMITS*
- *_o_SET WEB TIMEOUT*
- *_o_Web Context*

Presentación del servidor web

4D en modo local, 4D en modo remoto y 4D Server incluyen un servidor web que permite publicar bases 4D o todo tipo de página HTML en la Web. Las principales características del motor del servidor web de 4D son las siguientes:

- ***Fácil publicación***

Puede iniciar o detener la publicación de la base en la Web en cualquier momento. Para hacer esto, sólo necesita elegir un comando de menú o ejecutar un comando del lenguaje.

- ***Métodos de base dedicados***

Método de base On Web Authentication *y* ***Método base On Web Connection*** *son los puntos de entrada de las peticiones en el servidor web; pueden utilizarse para evaluar y direccionar todo tipo de petición.*

- ***Utilización de etiquetas y de URLs especiales***

El servidor web de 4D ofrece numerosos mecanismos que permiten la interacción con las acciones de los usuarios, en particular:
- pueden incluirse etiquetas especiales en las páginas web con el fin de iniciar el proceso por el servidor web en el momento en que se envían a los navegadores.
- los URLs especiales permiten llamar a 4D para ejecutar cualquier acción.
- estos URLs también pueden utilizarse como acciones de formulario para activar los procesos cuando el usuario envía formularios HTML.

- ***Seguridad de acceso***

Varias opciones de configuración automática le permiten otorgar autorizaciones de acceso específicas a los navegadores web o utilizar el sistema de contraseñas integrado de 4D. Puede definir un "Usuario web genérico" para simplificar la gestión de los accesos al interior de la base.
El ***Método de base On Web Authentication*** *permite evaluar toda petición antes de que sea procesada por el servidor web. Además, la capacidad de definir una carpeta raíz HTML por defecto permite restringir el acceso a los archivos en el disco. Por último, debe designar individualmente los métodos de proyecto que pueden ser ejecutados vía Web.*

- ***Conexiones SSL***

El servidor web 4D puede comunicarse con los navegadores web en modo seguro por medio del protocolo SSL (Secured Socket Layer). Este protocolo, compatible con la mayoría de los navegadores web, permite autenticar el emisor y receptor y garantiza la confidencialidad e integridad de la información intercambiada.

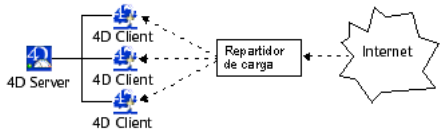
- ***Soporte extendido de los formatos de Internet***

El servidor web 4D es compatible HTTP/1.1 y soporta documentos XML y la tecnología WML (Wireless Markup Language).
El servidor web 4D también extiende el soporte de la compresión GZIP: al terminar una "negociación" entre el servidor y el cliente web, todos los intercambios pueden ser comprimidos potencialmente, para una mejora inmediata en el rendimiento.

- ***Operación simultánea de las bases de datos***
  - ***4D en modo local y la Web***
    Si publica una base 4D en la web utilizando 4D en modo local, puede simultáneamente:
    - Utilizar la base localmente con 4D
    - Conectarse a la base utilizando un navegador web.
- ***4D Server y la Web***
  Si publica una base 4D en la Web utilizando 4D Server, puede conectarse a la base 4D y utilizarla simultáneamente:
  - desde estaciones de trabajo 4D Remotas
  - desde navegadores Web.
- ***4D en modo remoto y la Web***
  Cuando una base 4D se publica en la Web con 4D Client, es posible conectarse a la base 4D y utilizarla simultáneamente:
  - vía equipos 4D remotos
  - vía los navegadores web. En este caso, si la base también está publicada con 4D Server, los navegadores web pueden conectarse a la base publicada vía cliente 4D o vía 4D Server. Además, esto permite administrar diferentes modos de acceso a los datos (público, administración, etc.).
  Los mecanismos básicos del servidor web de 4D se utilizan de una forma similar por 4D en modo remoto, con excepción del modo contextual. De hecho, no es posible utilizar el modo contextual con el servidor Web de 4D en modo remoto.
  De la misma forma, el funcionamiento de los comandos de lenguaje es generalmente idéntico, sin importar si el comando se ejecuta en 4D en modo local, 4D Server o 4D en modo remoto. El principio es que los comandos se aplican al sitio web del equipo en el cual se ejecutan. Debe administrar este principio utilizando los comandos ***Execute on server*** */* ***EXECUTE ON CLIENT****.*

- ***Repartición de la carga con clientes 4D****: como todo equipo 4D corriendo remotamente puede ser utilizado como servidor web, puede definir un sistema de servidor web dinámico con repartidor de carga. Esto ofrece muchas posibilidades de desarrollo, más particularmente:*
  - *la configuración de un sistema de repartición de carga para optimizar el rendimiento del servidor Web 4D: utilizando una replica del sitio web que está instalada en cada servidor web 4D Client, un repartidor de carga (hardware o*

software) enviará peticiones a los equipos cliente en función de su carga actual.



- *la configuración de un servidor web de tolerancia de falla: el sitio web 4D se replica en uno o más equipos 4D Client. En caso de falla de un servidor web 4D, otro lo reemplaza.*
- *la creación de vistas diferentes de los mismos datos, por ejemplo en función del origen de las peticiones. Dentro de una red empresarial, un servidor web en un equipo cliente 4D protegido puede servir las peticiones Intranet y otro servidor web de equipo cliente 4D, ubicado más allá del firewall, servirá las peticiones Internet.*
- *la repartición de las tareas entre los diferentes servidores web 4D en maquinas cliente: un servidor web 4D puede estar encargado de las peticiones SOAP, otro de las peticiones estándar, etc.*

Configuración del servidor web y gestión de conexiones

4D y 4D Server incluyen un servidor web (también llamado servidor HTTP) que permite publicar de manera transparente y dinámica los datos de sus bases en la Web.

Esta sección describe las etapas necesarias para el lanzamiento del servidor web y para la conexión de navegadores, así como los procesos de gestión de conexiones.

Condiciones de publicación de una base 4D en la web

Para poder lanzar el servidor HTTP de 4D o de 4D Server, debe disponer de los elementos descritos a continuación:

- *Una licencia "4D Web Application". Para mayor información, consulte la guía de instalación 4D.*
- *Las conexiones web se efectúan por medio del protocolo de red TCP/IP. Por consiguiente:*
  - *Debe tener el protocolo TCP/IP instalado y correctamente configurado en su equipo. Consulte la documentación de su ordenador o de su sistema operativo para mayor información sobre este punto.*
  - *Si quiere utilizar el protocolo SSL para sus conexiones, asegúrese de que los componentes necesarios se instalen correctamente (ver la sección **Utilizar el protocolo TLS (HTTPS)**).*
- *Una vez controlados y ajustados los puntos anteriores, debe iniciar el servidor web dentro de 4D. Este último punto se discute más adelante en esta sección.*

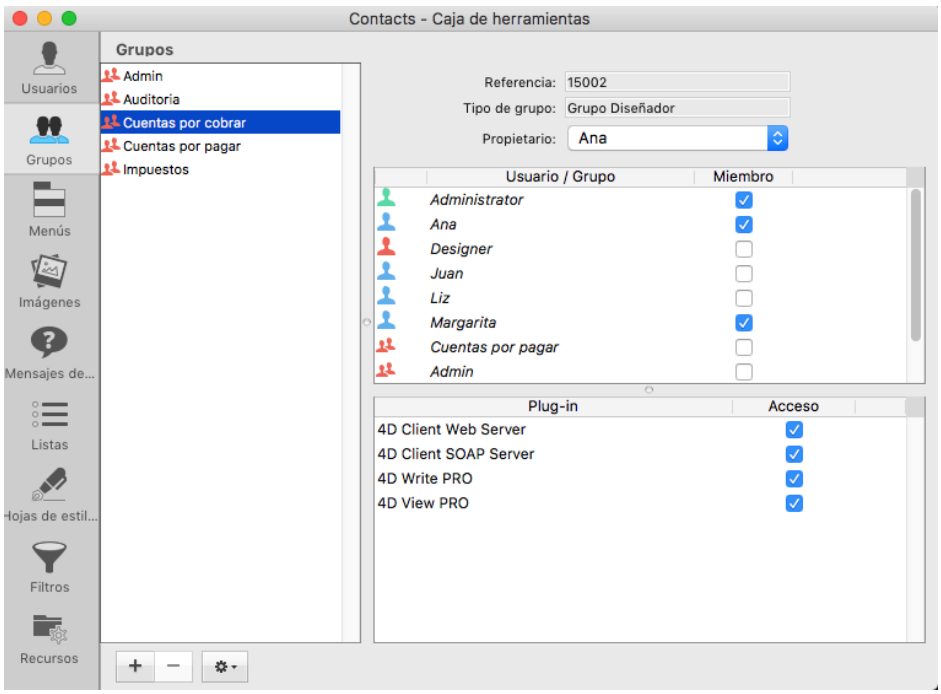
Autorización de publicación (4D en modo remoto)

Por defecto, todo equipo cliente 4D puede publicar en la Web la base a la cual está conectado. Sin embargo, puede controlar la posibilidad de publicación web de cada cliente 4D utilizando el sistema de contraseñas de 4D.

De hecho, las licencias web 4D son consideradas por 4D Server como licencias de plug-ins. Por lo tanto, de la misma forma que los plug-ins, debe restringir el uso de licencias Web Server a un grupo de usuarios específico.

*Para hacer esto, muestre la página **Grupos** en la Caja de herramientas utilizando 4D (debe disponer de las autorizaciones de acceso adecuadas para modificar estos parámetros).*

*Seleccione un grupo en la lista de la izquierda, luego seleccione la opción **Acceso** junto a la línea **4D Client Web Server** en el área de distribución de los plug-ins:*



Arriba: sólo los usuarios que pertenecen al grupo "4D Client Web Server" están autorizados a publicar su máquina 4D como servidor web.

HelperTool bajo Mac OS X

Bajo Mac OS X, la utilización de los puertos TCP/IP reservados a la publicación web (puertos 0 a 1023) requiere de privilegios de acceso específicos. Para poder utilizar estos puertos, 4D ofrece un programa utilitario llamado HelperTool. Cuando este programa esta instalado, recupera los privilegios de acceso y se encarga automáticamente de la apertura de los puertos web. Este mecanismo funciona con 4D (todos los modos), 4D Server y las aplicaciones ejecutables 4D Volume Desktop.

La aplicación HelperTool se incluye en el software 4D. La instalación se efectúa automáticamente durante la primera apertura de un puerto <1024 en la maquina. Se le informa al usuario que una herramienta se va a instalar y se le pide introducir un nombre y una contraseña de administrador de la máquina. Esta operación se realiza una sola vez.

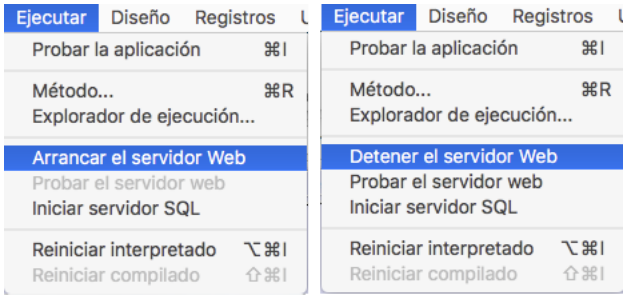
La aplicación se renombra "com.4D.HelperTool" y se instala en la carpeta "/Library/PrivilegedHelperTools/." Después de la secuencia inicial, el servidor web de 4D puede iniciarse y detenerse de manera transparente, sin importar la versión de 4D.

Iniciar el servidor web 4D

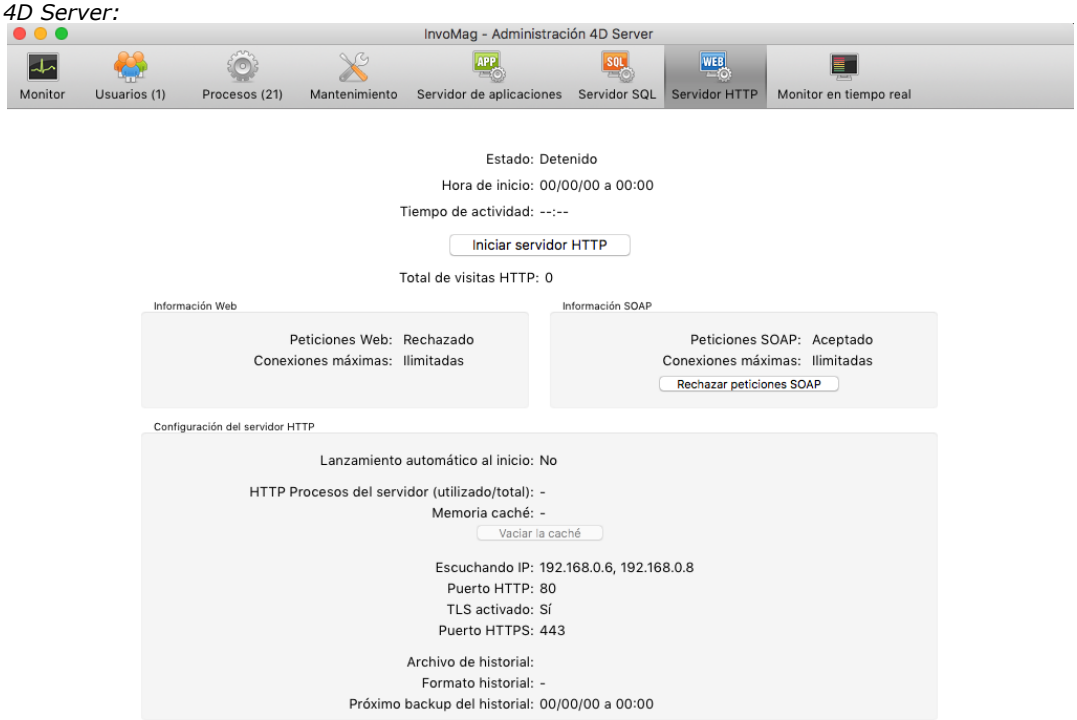
El servidor web 4D puede iniciarse de tres formas diferentes:

- *Por medio del menú Ejecutar de 4D o de la página Servidor HTTP (botón Iniciar servidor HTTP) en 4D Server. Estos comandos permiten iniciar y detener el servidor web en cualquier momento:*

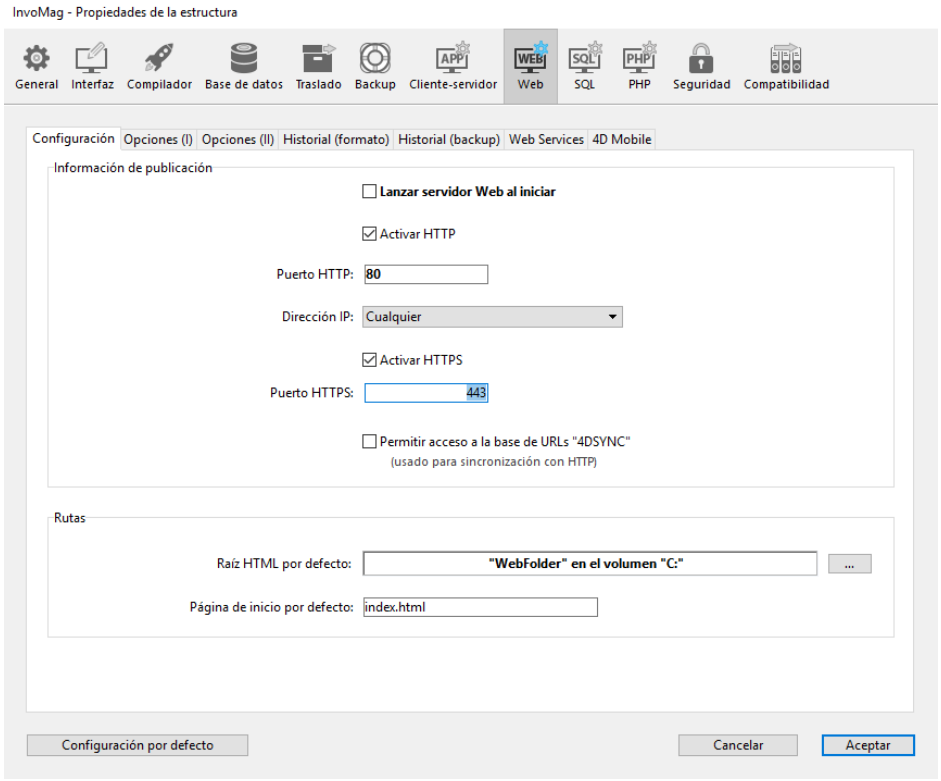
4D:



4D Server:



- *Para la publicación automática de la base en la Web cada vez que se abre, vaya a la página **Configuración** del tema **Web** de las Preferencias de la aplicación:*



*En la sección Publicación del servidor web, seleccione la casilla de selección **Publicar la base al iniciar**, luego haga clic en **Aceptar**. La base se publicará automáticamente como servidor web cada vez que la abra con 4D o 4D Server.*

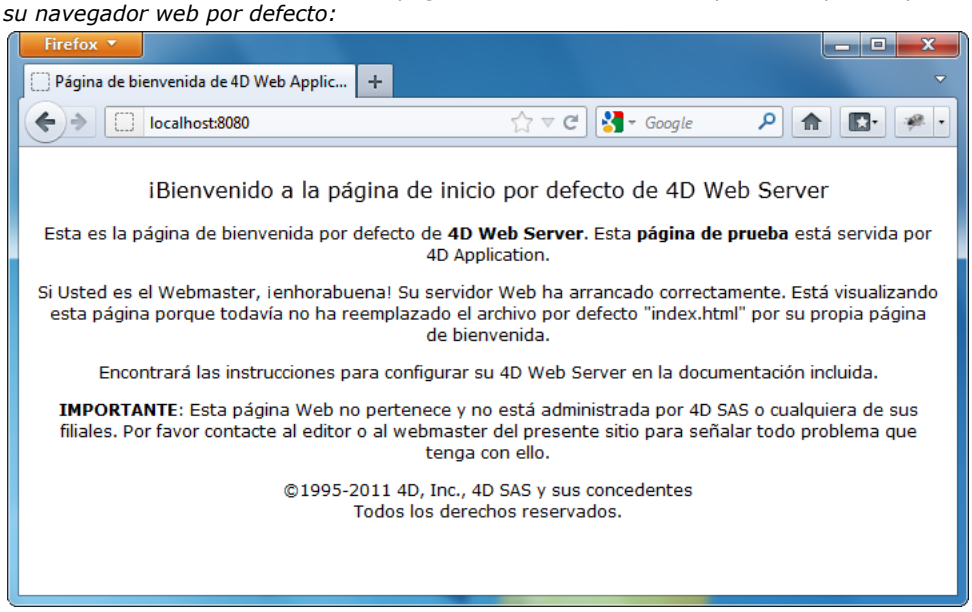
- *Por programación, llamando el comando **WEB START SERVER**.*

***Consejo:** No es necesario salir reabrir su base para iniciar o detener su publicación como servidor web. Puede interrumpir y reiniciar el servidor web tantas veces como sea necesario, utilizando el menú **Ejecutar**, del botón **Iniciar el servidor HTTP** o llamando a los comandos **WEB START SERVER** y **WEB STOP SERVER**.*

Probar el servidor web

*El comando **Probar servidor web** permite controlar el funcionamiento del servidor web integrado (4D únicamente). Este comando es accesible en el menú **Ejecutar** cuando se lanza el servidor web:*

Cuando selecciona este comando, la página de inicio del sitio web publicado por la aplicación 4D se muestra en una ventana de su navegador web por defecto:



Este comando permite verificar el funcionamiento del servidor web, la visualización de la página de inicio, etc. La página se llama utilizando el URL Localhost, que es el atajo estándar que designa la dirección IP del equipo en el cual se ejecuta el navegador. El comando tiene en cuenta el número de puerto TCP de publicación especificado en las Preferencias de la aplicación.

Conexión a una base 4D publicada en la Web

Una vez haya iniciado la publicación de una base 4D en la Web, puede conectarse utilizando un navegador web. Para hacer esto:

- *Si su sitio web tiene un nombre registrado (por ejemplo, " www.floresbonitas.com"), indique este nombre en el área Abrir, Dirección, o Ubicación de su navegador. Luego presione **Enter** para conectarse.*
- *Si su sitio web no tiene un nombre registrado, indique la dirección IP de su equipo (por ejemplo, 123.4.567.89) en el área Abrir, Dirección, o Ubicación de su navegador. Luego presione **Enter** para conectarse.*

En este momento, su navegador debe mostrar la página de inicio de su sitio web. Si ha publicado una base conservando la configuración estándar, debe obtener la página de inicio por defeco del servidor web de 4D. Esta página le permite probar la conexión y el funcionamiento del servidor.

También puede encontrarse con una de las siguientes situaciones:

1. La conexión falla y obtiene un mensaje del tipo "...el servidor no acepta las condiciones o puede estar ocupado...". En este caso, efectúe los siguientes controles:

- *Verifique que el nombre o la dirección IP que introdujo sea correcta.*
- *Verifique que 4D o 4D Server esté bien lanzado y que el servidor web se haya iniciado.*
- *Verifique que la base de datos esté bien configurada para ser publicada en el puerto TCP Web por defecto, es decir el 80 (ver la situación 4).*
- *Verifique que el protocolo de red TCP/IP esté configurado correctamente en el equipo servidor y en el equipo del navegador. Ambos equipos deben estar en la misma red y subred, o sus ruteadores deben estar correctamente configurados.*
- *Verifique las conexiones físicas.*
- *Si no prueba localmente su propio sitio, sino que trata de conectarse a una base web publicada en Internet o Intranet por alguien más, puede que finalmente el mensaje sea cierto: el servidor puede estar apagado u ocupado. De manera que intente nuevamente hasta que pueda conectarse, o contacte a su proveedor web.*

*2. Se establece la conexión, pero obtiene un error HTTP 404 "Archivo no encontrado". Esto significa que la página inicio del sitio no ha sido servida. En este caso, verifique que la página de inicio actual existe en la ubicación definida en las Propiedades de la base (ver la sección **Parámetros del servidor web**) o utilice el comando **WEB SET HOME PAGE.***

3. Se establece la conexión, pero NO obtiene la página web que estaba esperando. Esto puede ocurrir cuando tiene varios servidores web corriendo simultáneamente en el mimo equipo. Ejemplos:

- *usted lanzó una sola base Web 4D, en un sistema Windows que ya ejecuta su propio servidor web.*
- *usted está corriendo varias bases web 4D sobre el mismo equipo.*

*En este tipo de situación, necesita cambiar el puerto TCP en el cual se publica su base 4D Web se publica. Para hacer esto, consulte la sección **Parámetros del servidor web**.*

***Nota:** si su base está protegida por un sistema de contraseñas, podría tener que introducir un nombre de usuario y contraseña válidos (para mayor información, consulte la sección **Seguridad de las conexiones**).*

Procesos Web

Cada vez que un navegador web intenta conectarse a la base, la petición es manejada de la siguiente manera:

- *Primero, crea uno o varios procesos 4D locales llamados **Procesos de conexión web** para evaluar y administrar la conexión al navegador web.*
  *Estos procesos administran todas las peticiones HTTP. Se ejecutan rápidamente y luego se abortan o retrasan. Para optimizar el servidor web, una vez que se hace una petición, 4D congela esta "piscina" de procesos web por unos pocos segundos de manera que pueda reutilizarlos si es necesario cuando otra petición llega. Puede personalizar este comportamiento (tiempo de espera, número mínimo y máximo de procesos a conservar en la "piscina") utilizando el comando **SET DATABASE PARAMETER**.*
- *El proceso web se encarga del procesamiento de la petición y envía una respuesta (si es necesario) al navegador. El proceso temporal luego se aborta o retrasa (ver arriba).*

Soporte de IP v6

A partir de la versión 14, 4D soporta la notación de direcciones IPv6. Esto concierne a los siguientes servidores 4D, es decir:

- *el servidor web, así como el servidor SOAP,*
- *el servidor SQL.*

Nota: *para obtener más información acerca de IPv6, consulte la siguiente especificación: RFC 2460.*

El soporte de IPv6 es transparente para los usuarios y para los desarrolladores 4D: el programa acepta indiferentemente las conexiones IPv6 o IPv4 cuando la configuración "Dirección IP" del puerto de escucha del servidor es ***Todos*** *(ver* ***Dirección IP*** *(servidor HTTP) y* ***Preferencias de publicación del servidor SQL*** *(servidor SQL).*

Sin embargo, debe prestar atención a los siguientes puntos:

- ***Indicación de los números de puerto***
  La notación IPv6 usa dos puntos (:), la adición de los números de puerto puede traer cierta confusión, por ejemplo:

  ```
  2001:0DB8::85a3:0:ac1f:8001 // dirección IPv6
  2001:0DB8::85a3:0:ac1f:8001:8081 // dirección IPv6 puerto 8081
  ```

  Para evitar esta confusión, se recomienda utilizar la notación [ ] cuando se combina una dirección IPv6 con un número de puerto, por ejemplo:

  ```
  [2001:0DB8::85a3:0:ac1f:8001]:8081 //dirección IPv6 puerto 8081
  ```

- ***No detección de la ocupación del puerto TCP***
  A diferencia de versiones anteriores de 4D, cuando el servidor esta configurado para responder a "todas" las direcciones IP con 4D v14, si el puerto TCP está siendo utilizado por otra aplicación, ya no se indica cuando se inicia el servidor. De hecho, el servidor 4D no detecta ningún error en este caso debido a que el puerto permanece libre bajo la dirección IPv6. Sin embargo, no es posible acceder a ella mediante la dirección IPv4 de la máquina, ni por medio de la dirección local: 127.0.0.1.
  Si su servidor 4D no parece responder en el puerto definido, puede probar la dirección [::1] en el servidor (equivalente bajo IPv6 a 127.0.0.1, añada :numPort para probar un número de puerto diferente al número por defecto). Si 4D responde, es probable que otra aplicación esté usando el puerto en IPv4.
- ***Direcciones IPv6 basados en IPv4***
  Para estandarizar el procesamiento, 4D ofrece una representación híbrida estándar de direcciones IPv4 en IPv6. Estas direcciones se escriben con un prefijo de 96 bits en formato IPv6, seguido de 32 bits escritos en la notación decimal de puntos de IPv4. Por ejemplo, ::ffff:192.168.2.34 representa la dirección IPv4 192.168.2.34.

Seguridad de las conexiones

La seguridad de su servidor web 4D está basada en los siguientes elementos:

- *La combinación del sistema de gestión de contraseñas (modo BASIC o modo DIGEST) y del **Método de base On Web Authentication**,*
- *La definición de un Usuario web genérico,*
- *La definición de una carpeta Raíz HTML por defecto,*
- *La definición de la propiedad "Disponible vía las etiquetas y los URLs 4D (4D ACTION.)" para cada método proyecto de la base.*
- *La opción para el soporte específico de peticiones de sincronización vía HTTP.*

Notas:

- *la seguridad de la conexión misma puede ser administrada por el protocolo SSL. Para mayor información, consulte la sección [#title id="777"/].*
- *para una descripción general de las funcionalidades de seguridad de 4D, consulte la guía de seguridad 4D.*

Sistema de gestión de contraseñas para el acceso Web

Modo BASIC y Modo DIGEST

Puede definir en las preferencias de la base, el sistema de control de acceso que quiere aplicar a su servidor web. Se ofrecen dos modos de autenticación: Modo BASIC y modo DIGEST.
El modo autenticación concierne a la manera como se colecta y procesa la información relativa al nombre de usuario y contraseña.

- *En modo BASIC, el nombre y la contraseña introducida por el usuario son enviadas sin encriptar en las peticiones HTTP. Este principio no asegura una seguridad total al sistema ya que la información puede ser interceptada y utilizada por terceros.*

- *El modo DIGEST ofrece un nivel mayor de seguridad ya que la información de autenticación es procesada por un proceso unidireccional llamado dispersión (hashing) que hace que su contenido sea imposible de descifrar.*

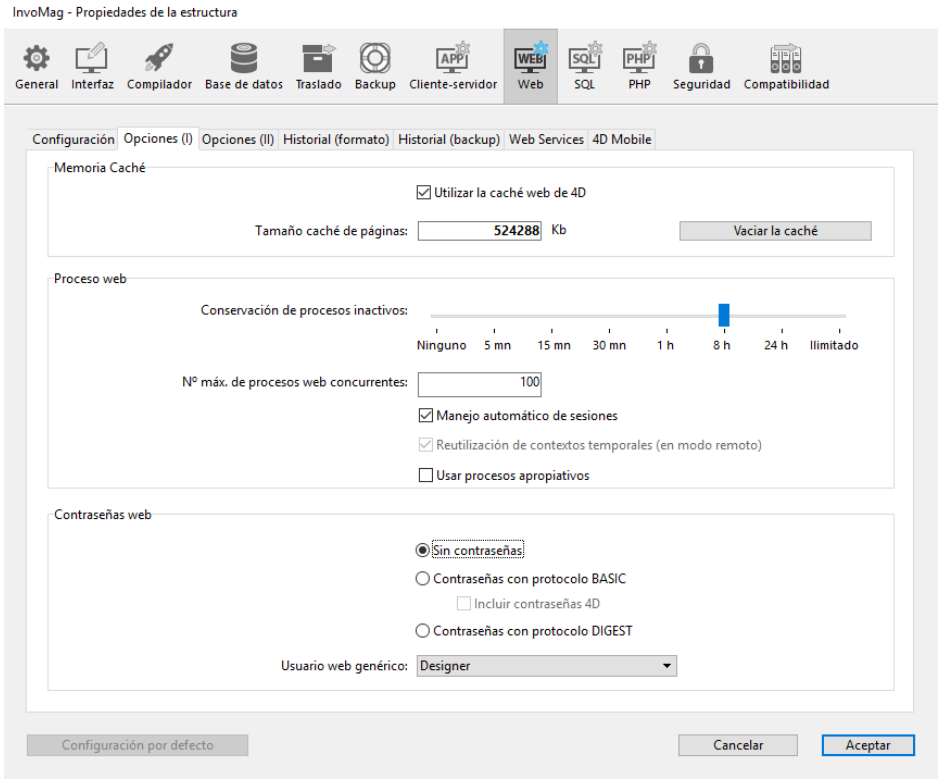
Para el usuario, el uso de un modo de autenticación u otro es transparente.

Notas:

- *Por razones de compatibilidad, el modo de autenticación BASIC se utiliza por defecto en las bases 4D convertidas a versión 11 (si la opción "Utilizar contraseñas" fue seleccionada en la versión anterior). Debe activar explícitamente el modo Digest.*

- *La autenticación Digest es una función de HTTP1.1 y no es soportada por todos los navegadores. Por ejemplo, sólo las versiones 5.0 y superiores de Microsoft Internet Explorer aceptan este modo. Si un navegador no soporta esta funcionalidad envía una petición a un servidor web cuando se activa la autenticación Digest, el servidor rechazará la petición y devolverá un mensaje de error al navegador.*

*La elección del modo de autenticación se efectúa en la página **Web/Opciones (I)** de la caja de diálogo de Propiedades de la base:*



En el área "Contraseñas web", hay tres opciones disponibles:

- ***Sin contraseñas**: ninguna autenticación se lleva acabo para la conexión al servidor web. En este caso:*

*- Si existe el **Método de base On Web Authentication**, se ejecuta y además de $1 y $2, sólo las direcciones IP del navegador y del servidor ($3 y $4) son suministradas, el nombre de usuario y la contraseña ($5 y $6) están vacíos. En este caso, puede filtrar las conexiones en función de la dirección IP del navegador y/o de la dirección IP pedida del servidor.*
*- Si el **Método de base On Web Authentication** no existe, las conexiones son aceptadas automáticamente.*

- ***Contraseñas con protocolo BASIC**: autenticación estándar en modo BASIC. Cuando un usuario se conecta al servidor, aparece una caja de diálogo en su navegador permitiéndole introducir su nombre y su contraseña. Estos dos valores se envían al **Método de base On Web Authentication** junto con los otros parámetros de la conexión (dirección y puerto IP, URL...) de manera que usted los pueda procesar.*
*Este modo ofrece acceso a la opción **Incluir contraseñas 4D** que permite utilizar, en lugar de o en adición a su propio sistema de contraseñas, el sistema 4D de contraseñas de la base (definido en 4D).*

- ***Contraseñas con protocolo DIGEST**: autenticación en modo DIGEST. Como en modo BASIC, los usuarios deben introducir su nombre y contraseña cuando se conectan. Estos dos valores son enviados encriptados al **Método de base On Web Authentication** con los otros parámetros de la conexión. Debe autenticar un usuario utilizando el comando **Ventana de resultados**.*

Notas:

- *Debe reiniciar el servidor web para que las modificaciones efectuadas a estos parámetros sean tenidas en cuenta*
- *Con el servidor web de 4D Client, recuerde que todos los sitios publicados por los equipos clientes 4D compartirán la misma tabla de usuarios. La validación de los usuarios/contraseñas es llevada a cabo por la aplicación 4D Server.*

Modo BASIC: Combinación de contraseñas y del método de base On Web Authentication

Si utiliza el modo BASIC, el sistema de filtro de las conexiones al servidor web de 4D depende de la combinación de dos parámetros:

- *Las opciones de contraseñas web en la caja de diálogo de Propiedades de la base,*
- *La existencia del **Método de base On Web Authentication**.*

Estas son las diferentes posibilidades de control de las conexiones:
La opción "Contraseñas con protocolo BASIC" está seleccionada y la opción "Incluir contraseñas 4D" no está seleccionada.

- *Si existe el **Método de base On Web Authentication**, se ejecuta y todos sus parámetros son dados. Puede filtrar más precisamente las conexiones de acuerdo al nombre de usuario, contraseña, y/o a las direcciones IP del navegador y del servidor web.*

- *Si el **Método de base On Web Authentication** no existe, la conexión se rechaza automáticamente y se envía un mensaje al navegador indicando que el método de autenticación no existe.*

***Nota:** si el nombre de usuario enviado es una cadena vacía y si el **Método de base On Web Authentication** no existe, se envía al navegador una caja de diálogo de petición de contraseña.*
Las opciones "Contraseñas con protocolo BASIC" e "Incluir contraseñas 4D" están seleccionadas.

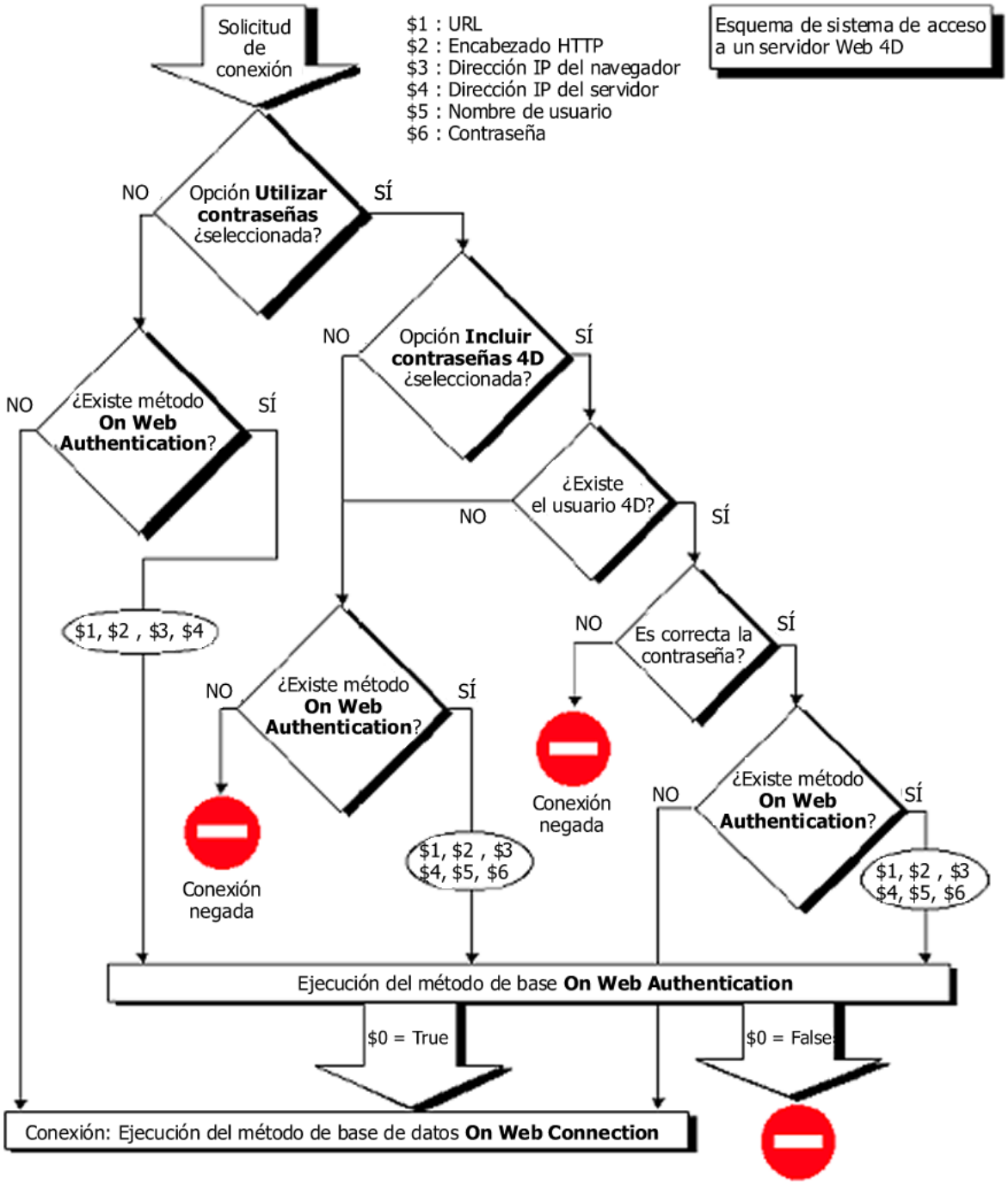
- *Si el nombre de usuario enviado por el navegador existe en la tabla de usuarios 4D y la contraseña es correcta, la conexión es aceptada. Si la contraseña es incorrecta, la conexión es rechazada.*

- *Si el nombre de usuario enviado por el navegador no existe en 4D, dos resultados son posibles:*

*- Si existe el **Método de base On Web Authentication**, se devuelven los parámetros $1, $2, $3, $4, $5, y $6. Por lo tanto puede filtrar las conexiones de acuerdo con el nombre de usuario, contraseña, y/o las direcciones IP del navegador y del servidor web.*
*- Si el **Método de base On Web Authentication** no existe, la conexión se rechaza.*

Modo DIGEST

A diferencia del modo BASIC, el modo DIGEST no es compatible con las contraseñas 4D estándar: no es posible utilizar las contraseñas 4D como identificadores web. La opción "Incluir contraseñas 4D" está gris cuando se selecciona este modo. Los identificadores de los usuarios Web deben ser administrados de manera personalizada (por ejemplo, vía una tabla).
*Cuando el modo DIGEST está activo, el parámetro $6 (contraseña) siempre se devuelve vacío en el **Método de base On Web Authentication**. De hecho, cuando se utiliza este modo, esta información no pasa por la red como texto claro (no encriptado). Por lo tanto es imperativo en este caso evaluar la petición de conexión con la ayuda del comando **WEB Validate digest**.*
El funcionamiento del sistema de acceso al servidor web 4D se resume en el siguiente diagrama:



Acerca de los robots (nota de seguridad)

Algunos robots (motores de búsqueda, arañas...) recorren los servidores Web y las páginas estáticas. Si quiere que los robots puedan acceder a todo su sitio, puede definir cuáles URLs no pueden acceder.
Para esto, ponga el archivo ROBOTS.TXT en la raíz del servidor Web. Este archivo debe estar estructurado de la siguiente manera:

User-Agent: <nombre>
Disallow: <URL> o <comienzo del URL>

Por ejemplo:

*User-Agent: **
Disallow: /4D

Disallow: /%23%23
Disallow: /GIFS/

*"User-Agent: *" significa que todos los robots están afectados.*
"Disallow: /4D" significa que los robots no deben acceder a los URLs que comienzan por /4D.
"Disallow: /%23%23" significa que los robots no deben acceder a los URLs que comienzan por /%23%23.
"Disallow: /GIFS/' significa que los robots no deben acceder a la carpeta /GIFS/ ni a las subcarpetas.

Otro ejemplo:

*User-Agent: **
Disallow: /

En este caso, los robots no tienen acceso a todo el sitio.

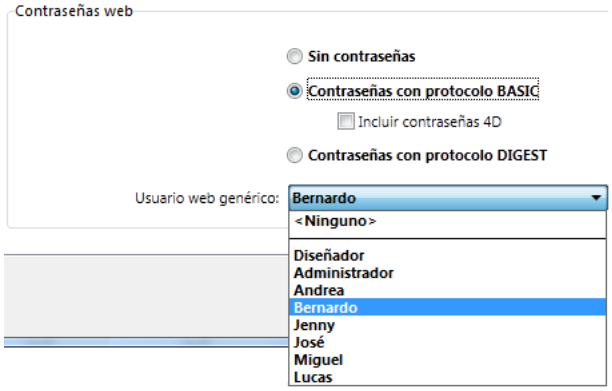
Usuario web genérico

Puede designar un usuario, previamente definido en la tabla de contraseñas de 4D, como "Usuario Web Genérico." En este caso, cada navegador que se conecta a la base puede utilizar las autorizaciones de acceso y las restricciones asociadas con este usuario. De esta forma puede controlar fácilmente el acceso de los navegadores a las diferentes partes de la base.

***Nota:** no hay que confundir esta opción, la cual permite restringir los accesos de los navegadores a las diferentes partes de la aplicación (métodos, formularios, etc.), con el sistema de control de conexiones al servidor web, administrado por el sistema de contraseñas y el **Método de base On Web Authentication**.*

Para definir un Usuario web genérico:

1. *En modo Diseño, cree al menos un usuario con el editor de usuarios de la caja de herramientas.*
   Si quiere puede asociar una contraseña con el usuario.
2. *En los diferentes editores de 4D, autorice o restrinja el acceso a este usuario.*
3. *En la caja de diálogo de Propiedades, elija el tema **Web**, página **Opciones (I)**.*
   *El área "Contraseñas Web" contiene la lista desplegable **Usuario web genérico**. Por defecto, el Usuario web genérico es el Diseñador y los navegadores tienen acceso completo a*
   todas las partes de la base.
4. *Elija un usuario en la lista desplegable y valide la caja de diálogo.*



Todos los navegadores web autorizados a conectarse a la base se beneficiarán de las autorizaciones de acceso asociadas al Usuario web genérico (excepto cuando el modo BASIC y la opción "Incluir contraseñas 4D" están seleccionados y el usuario que se conecta no existe en la tabla de contraseñas 4D, ver a continuación).

Interacción con el protocolo BASIC

La opción "Contraseñas con protocolo BASIC" no influye en cómo funciona el Usuario Web genérico. Sin importar el estado de esta opción, los privilegios y las restricciones de acceso asociados al "Usuario Web genérico" se aplicarán a todos los navegadores Web que están autorizados para conectarse a la base.

Sin embargo, cuando la opción "Incluir contraseñas 4D" está seleccionada pueden presentarse dos posibles resultados:

- *El nombre y la contraseña del usuario no existen en la tabla de contraseñas de 4D. En este caso, si la conexión ha sido aceptada por el **Método de base On Web Authentication**, los derechos de acceso del usuario Web genérico serán aplicados al navegador.*
- *Si el nombre de usuario y contraseña existen la tabla de contraseñas de 4D, el parámetro "Usuario Web genérico" se ignora. El usuario se conecta con sus propios derechos de acceso.*

Raíz HTML por defecto

Esta opción de las Propiedades de la base permite definir la carpeta en la cual 4D buscará las páginas HTML estáticas y semidinámicas, las imágenes, etc., a enviar a los navegadores.

*Además, la carpeta raíz HTML define, en el disco duro del servidor web, el nivel jerárquico sobre el cual los archivos no serán accesibles. Esta restricción de acceso aplica a los URLs enviados a los navegadores web como también a los comandos del servidor web tal como **WEB SEND FILE**. Si un URL enviado a la base por un navegador o si un comando 4D intenta acceder a un archivo ubicado sobre la carpeta raíz HTML, se devuelve un error indicando que el archivo no ha sido encontrado.*

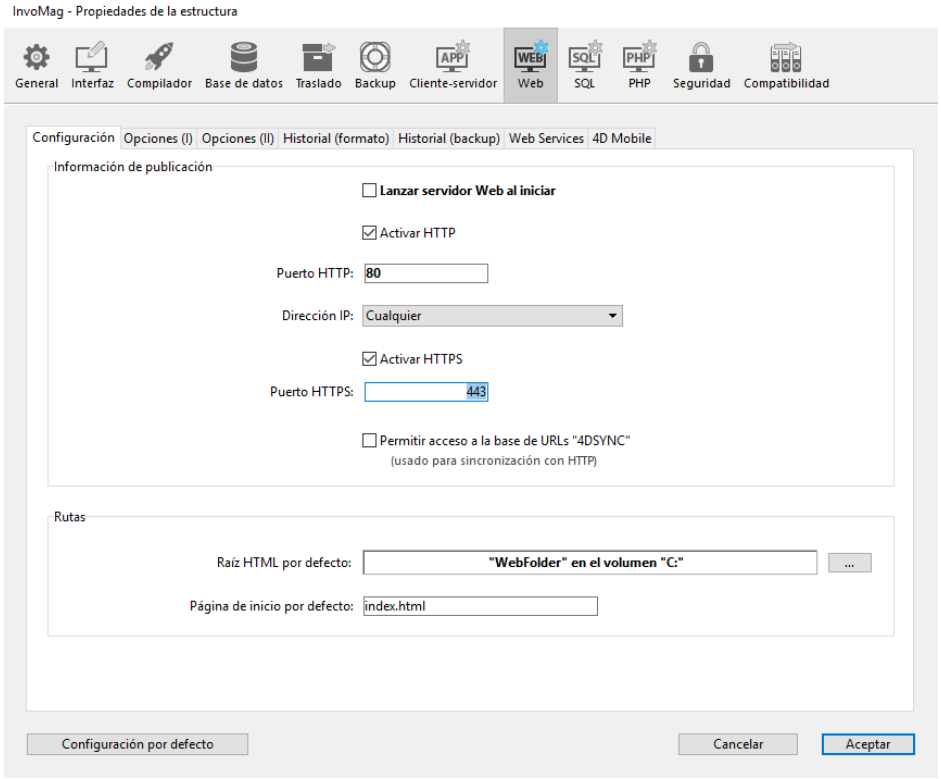
*Por defecto, 4D define una carpeta raíz HTML llamada **WebFolder**. Si esta carpeta no existe, la carpeta raíz HTML se crea en el disco en el momento en que se lanza el servidor web por primera vez.*

Si conserva la ubicación por defecto, la carpeta raíz se crea:

- *con 4D in modo local y 4D Server, al mismo nivel que el archivo de estructura de la base.*
  ***Nota:** como parte de una aplicación compilada y fusionada, el archivo de estructura se ubica en la subcarpeta **Database**.*

- *con 4D en modo remoto, en la carpeta local de la base 4D (ver el comando **Get 4D folder**).*

*Puede modificar el nombre y la ubicación de la carpeta raíz HTML por defecto en la caja de diálogo de Propiedades (tema **Web**, página **Configuración**):*



En el área "Raíz HTML por defecto", introduzca la nueva ruta de acceso de la carpeta que quiere utilizar.

La ruta de acceso introducida en esta caja de diálogo es relativa: está establecida a partir de la carpeta que contiene la estructura de la base (4D en modo local o 4D Server) o la carpeta que contiene la aplicación o el software 4D (4D in modo remoto).

Con el fin de asegurar la compatibilidad multiplataforma de sus bases, el servidor Web 4D utiliza convenciones de escritura particulares para describir rutas de acceso. Las reglas de sintaxis son las siguientes:

- *Las carpetas están separadas por una barra oblicua ("/")*
- *La ruta de acceso no debe terminar con una barra oblicua ("/")*
- *Para "subir" un nivel en la jerarquía de las carpetas, introduzca ".." (dos puntos) antes del nombre de la carpeta*
- *La ruta de acceso no debe comenzar con una barra oblicua ("/") (excepto si quiere que la carpeta raíz HTML sea la carpeta de la base o de 4D Client, ver a continuación).*

Por ejemplo, si quiere que la carpeta raíz HTML sea la subcarpeta "Web", ubicada en la carpeta "Base4D", introduzca Base4D/Web.

Si quiere que la carpeta raíz HTML sea la carpeta de la base o de 4D Client, pero que el acceso a las carpetas de los niveles superiores esté prohibido, introduzca "/" en el área. Para que el acceso a los volúmenes sea total, deje el área "Raíz HTML por defecto" vacía.

***Advertencia:** si no define ninguna carpeta raíz HTML por defecto, se utilizará la carpeta que contiene el archivo de estructura de la base o de la aplicación 4D Client. **Tenga cuidado porque en este caso, no hay restricciones de acceso** (los usuarios pueden acceder a todos los volúmenes).*

Notas:

- *Cuando la carpeta raíz HTML se modifica en las Propiedades de la base, la caché se borra para no conservar archivos cuyo acceso esté restringido.*
- *También es posible definir dinámicamente la carpeta raíz HTML utilizando el comando **WEB SET ROOT FOLDER**. En este caso, la modificación aplica a todos los procesos web actuales para la sesión de trabajo. La caché de las páginas HTML se borra.*

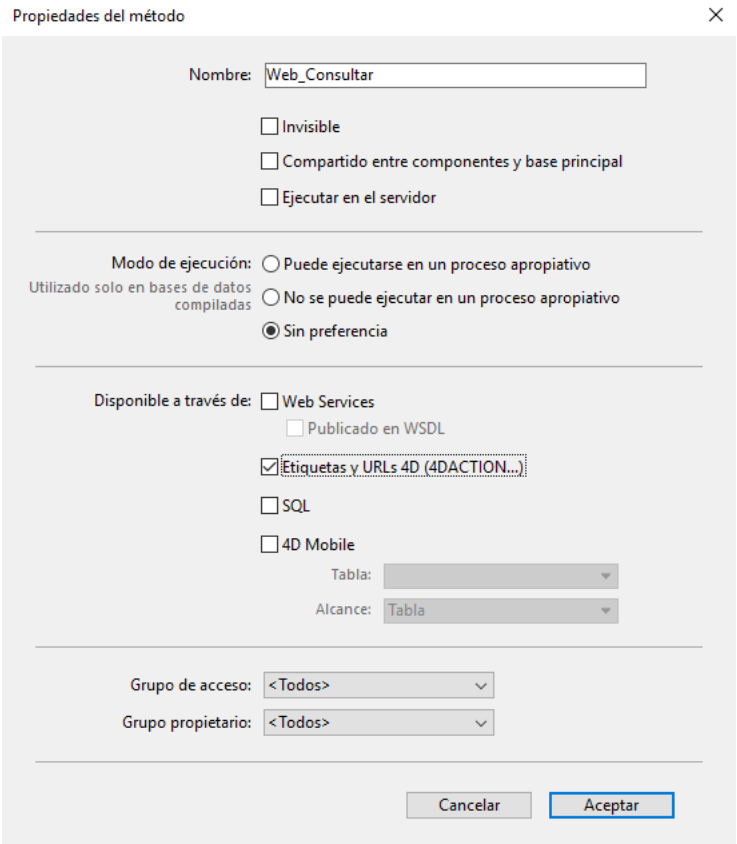
Disponible para las etiquetas y los URLS 4D

*El URL especial 4DACTION, las etiquetas 4DSCRIPT, 4DEVAL, 4DTEXT, 4DHTML (así como también las antiguas etiquetas 4DVAR y 4DHTMLVAR) permiten activar la ejecución de todo método proyecto de una base 4D publicada en la Web. Por ejemplo, la petición http://www.server.com/4DACTION/Borrar_Todo provoca la ejecución del método proyecto **Borrar_Todo**, si existe.*

Este mecanismo presenta un riesgo para la seguridad de la base, en particular si un internauta intencionalmente o por error dispara un método no destinado para ejecución vía Web. Puede evitar este riesgo de tres formas:

- *Restringiendo el acceso a los métodos proyecto utilizando el sistema de contraseñas 4D. Inconvenientes: este sistema necesita utilizar las contraseñas 4D y prohíbe todo tipo de ejecución del método (incluyendo utilizando las etiquetas HTML).*
- *Filtrando los métodos llamados vía los URLS utilizando el **Método de base On Web Authentication**. Inconvenientes: si la base incluye un gran número de métodos, este sistema puede ser difícil de administrar.*

- *Usar la opción **Etiquetas y los URLs 4D (4DACTION...)** mostrada en la caja de diálogo de Propiedades de los métodos proyecto:*



Esta opción permite designar individualmente cada método proyecto que puede llamarse utilizando el URL especial 4DACTION y las etiquetas 4DSCRIPT, 4DEVAL, 4DTEXT y 4DHTML (así como también 4DVAR y 4DHTMLVAR). Cuando está deseleccionada, el método proyecto concerniente no puede ejecutarse utilizando una petición HTTP que contenga un URL o etiqueta especial. Por el contrario, puede ejecutarse utilizando otro tipo de llamadas (fórmulas, otros métodos, etc.).

Esta opción está deseleccionada por defecto al crear una base. Los métodos que pueden ejecutarse utilizando el URL 4DACTION y las etiquetas 4D deben indicarse expresamente.

En el Explorador, los métodos proyecto con esta propiedad tienen un icono específico:

Autorizar el acceso a la base de datos vía los URLs 4DSYNC

*Esta opción de la página "Web/Configuración" de las Propiedades de la base permite controlar el soporte de peticiones que contienen los URLs /4DSYNC. Estos URLs se utilizan para la sincronización de datos a través de HTTP (para obtener más información sobre este mecanismo, consulte el párrafo **URL 4DSYNC/**).*

Esta opción habilita o deshabilita el procesamiento especifico de las peticiones que contienen /4DSYNC:

- *Cuando no está seleccionada, las peticiones /4DSYNC son considerados como peticiones estándar y no permiten el procesamiento específico (el uso de una petición de sincronización provoca el envío de la respuesta tipo "404 - recurso no disponible").*
- *Cuando se activa, el mecanismo de sincronización está activado; las peticiones /4DSYNC se consideran como peticiones especiales y son procesadas por el servidor HTTP de 4D.*

Por defecto:

- *esta opción no está seleccionada en bases de datos creadas con 4D a partir de la versión 13.*
- *esta opción está seleccionada en las bases de datos convertidas de una versión anterior a 4D v13, razones de compatibilidad. Le recomendamos que la desactive si su aplicación no utiliza la función de replicación HTTP.*

El alcance de esta opción es local a la aplicación y el servidor web se debe reiniciar para tenerla en cuenta.

Método de base On Web Authentication

Descripción

*El **Método de base On Web Authentication**está a cargo de administrar el acceso al motor del servidor web. Es llamado automáticamente por 4D o 4D Server cuando una petición de un navegador web requiere la ejecución de un método 4D en el servidor (llamada de un método vía un URL 4DACTION o una etiqueta 4DSCRIPT, etc.).*

Este método recibe seis parámetros de tipo Texto, pasados por 4D: $1, $2, $3, $4, $5, y $6 y devuelve un booleano, $0. La descripción de estos parámetros es la siguiente:

Parámetros	Tipo	Descripción
$1	Texto	URL
$2	Texto	Encabezado + Cuerpo HTTP (32 KB máximo)
$3	Texto	Dirección IP del navegador
$4	Texto	Dirección IP que llama al servidor
$5	Texto	Nombre del usuario
$6	Texto	Contraseña
$0	Booleano	True = petición aceptada, False = petición rechazada

Debe declarar estos parámetros de esta forma:

```
` Método de base On Web Authentication

C_TEXT($1;$2;$3;$4;$5;$6)
C_BOOLEAN($0)

` Código para el método
```

***Nota:** todos los parámetros del **Método de base On Web Authentication**no se llenarán. La información recibida por el método de base depende las opciones que haya seleccionado previamente en la caja de diálogo de Propiedades de la base. Consulte la sección **Seguridad de las conexiones**).*

- ***URL***

El primer parámetro ($1) es el URL introducido por el usuario en el área ubicación de su navegador web, menos la dirección local.

Tomemos el ejemplo de una conexión de Intranet. Supongamos que la dirección IP de su equipo servidor web 4D es 123.4.567.89. La siguiente tabla muestra los valores de $1 dependiendo del URL introducido en el navegador web:

URL introducido en el navegador web	Valor del parámetro $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clientes	/Clientes
http://123.4.567.89/Clientes	/Clientes
http://123.4.567.89/Clientes/Añadir	/Clientes/Añadir
123.4.567.89/Hacer_esto/Si_OK/Hacer_eso	/Hacer_esto/Si_OK/Hacer_eso

- ***Encabezado y cuerpo de la petición HTTP***

*El segundo parámetro ($2) es el encabezado y el cuerpo de la petición HTTP enviada por el navegador web. Note que esta información se pasa al **Método de base On Web Authentication** tal como está. El contenido varía en función del tipo de navegador web que esté intentando la conexión. Si su aplicación manipula esta información, es su decisión si analiza el encabezado y el cuerpo.*

Notas:

- *Por razones de rendimiento, el tamaño de los datos que transita vía el parámetro $2 no debe superar los 32 KB. De lo contrario serán truncados por el servidor HTTP de 4D.*
- *Para mayor información sobre este parámetro, consulte la descripción del **Método base On Web Connection**.*
- ***Dirección IP del navegador***
  El tercer parámetro $3 recibe la dirección IP del equipo navegador. Esta información permite distinguir entre las conexiones de Intranet e Internet.
  ***Nota:** 4D devuelve las direcciones IPv4 en un formato híbrido IPv6 escritos con un prefijo de 96 bits, por ejemplo ::ffff:192.168.2.34 para la dirección IPv4 192.168.2.34. Para mayor información, consulte la sección **Soporte de IP v6**.*
- ***Dirección IP para llamar al servidor***
  *El cuarto parámetro $4 recibe la dirección IP utilizada para llamar al servidor Web. 4D a partir de la versión 6.5 autoriza el multi-homing, permitiendo explotar equipos con más de una dirección IP. Para mayor información, consulte la sección **QR DELETE COLUMN**.*
- ***Nombre del usuario y contraseña***
  *Los parámetros $5 y $6 reciben el nombre de usuario y contraseña introducidos por el usuario en la caja de diálogo estándar de identificación mostrada por el navegador. Esta caja de diálogo aparece para cada conexión, si una opción de gestión de contraseñas ha sido seleccionada en las Propiedades de la base (ver la sección **Seguridad de las conexiones**).*

***Nota:** si el nombre de usuario enviado por el navegador existe en 4D, el parámetro $6 (la contraseña del usuario) no se devuelve por razones de seguridad.*

- ***Parámetro $0***

*El **Método de base On Web Authentication** devuelve un booleano en $0:*

  - *Si $0 es **True**, la conexión es aceptada.*
  - *Si $0 es **False**, la conexión es rechazada.*

*El **Método base On Web Connection** sólo se ejecuta si la conexión ha sido aceptada por **On Web Authentication**.*
***Advertencia:** si no se pasa ningún valor en $0 o si $0 no se define en el **Método de base On Web Authentication**, la conexión se considerará como aceptada y se ejecuta el **Método base On Web ConnectionIs Windows**.*
Notas:

- *No llame elementos de interfaz en el **Método de base On Web Authentication** (**ALERT**, **DIALOG**, etc.) porque de lo contrario su ejecución se interrumpirá y la conexión será rechazada. Lo mismo sucede si se presenta un error durante su proceso.*
- *Es posible evitar la ejecución por 4DACTION o 4DSCRIPT de cada método de proyecto con la ayuda de la opción "Disponible vía las etiquetas HTML y URLs (4DACTION...) en la caja de diálogo de las Propiedades de los métodos. Para mayor información sobre este punto, consulte la sección **Seguridad de las conexiones**.*

Llamadas del método base On Web Authentication

*El **Método de base On Web Authentication** se llama automáticamente, sin importar el modo, cuando una petición o proceso requiere la ejecución de un método 4D. También se llama cuando el servidor web recibe un URL estático inválido (por ejemplo, si la página estática solicitada no existe).*
*Por lo tanto el **Método de base On Web Authentication** se llama en los siguientes casos:*

- *cuando 4D recibe un URL que comienza por 4DACTION/*
- *cuando 4D recibe un URL que comienza por 4DCGI/*
- *cuando 4D recibe un URL que comienza por 4DSYNC/*
- *cuando 4D recibe un URL solicitando una página estática que no existe*
- *cuando 4D recibe un URL de acceso a la raíz y no se ha definido una página de inicio en las propiedades de la base o por medio del comando **WEB SET HOME PAGE***
- *cuando 4D procesa una etiqueta 4DSCRIPT en una página semidinámica*
- *cuando 4D procesa una etiqueta 4DLOOP basada en un método en una página semidinámica.*

***Nota de compatibilidad:** el método base también se llama cuando 4D recibe un URL que comienza por 4DMETHOD/. Este URL es obsoleto y sólo se conserva por razones de compatibilidad.*
*Note que el **Método de base On Web Authentication** NO se llama cuando el servidor recibe un URL solicitando una página estática válida.*

Ejemplo 1

*Ejemplo del **Método de base On Web Authentication** en modo BASIC:*

```
  `Método de base On Web Authentication
C_TEXT($5;$6;$3;$4)
C_TEXT($usuario;$contraseña;$IPNavegador;$IPServidor)
C_BOOLEAN($4Dusuario)
ARRAY TEXT($usuarios;0)
ARRAY LONGINT($nums;0)
C_LONGINT($upos)
C_BOOLEAN($0)

$0:=False

$usuario:=$5
$contraseña:=$6
$IPNavegador:=$3
$IPServidor:=$4

  `Por razones de seguridad, rechazar nombres que contengan @
If(WithWildcard($usuario)|WithWildcard($contraseña))
   $0:=False
  `El método WithWildcard se describe a continuación
Else
  `Verificar para ver si es un usuario 4D
   GET USER LIST($usuarios;$nums)
   $upos:=Find in array($usuarios;$usuario)
   If($upos >0)
      $usuario4D:=Not(Is user deleted($nums{$upos}))
   Else
      $usuario4D:=False
   End if

   If(Not($usuario4D))
```

```
   `No es un usuario definido en 4D, buscar en la tabla de usuarios Web
      QUERY([UsuariosWeb];[UsuariosWeb]usuario=$usuario;*)
      QUERY([UsuariosWeb]; & [UsuariosWeb]contraseña=$contraseña)
      $0:=(Records in selection([UsuariosWeb])=1)
   Else
      $0:=True
   End if
End if
   `¿Esta es una conexión de intranet?
If(Substring($IPNavegador;1;7)#"192.100.")
   $0:=False
End if
```

Ejemplo 2

*Ejemplo de **Método de base On Web Authentication** en modo DIGEST:*

```
   // Método base On Web Authentication
C_TEXT($1;$2;$5;$6;$3;$4)
C_TEXT($user)
C_BOOLEAN($0)
$0:=False
$user:=$5
   // Por razones de seguridad, rechazar los nombres que contengan @
If(WithWildcard($user))
   $0:=False
   // El método <span class="rte4d_met">WithWildcard</span> se describe a continuación
Else
   QUERY([WebUsers];[WebUsers]User=$user)
   If(OK=1)
      $0:=WEB Validate digest($user;[WebUsers]password)
   Else
      $0:=False  // Usuario inexistente
   End if
End if
```

El método WithWildcard es:

```
   // Método WithWildcard
   // WithWildcard ( String ) -> Booleano
   // WithWildcard ( Name ) -> Contiene un carácter arroba


C_LONGINT($i)
C_BOOLEAN($0)
C_TEXT($1)

$0:=False
For($i;1;Length($1))
   If(Character code(Substring($1;$i;1))=Character code("@"))
      $0:=True
   End if
End for
```

Método base On Web Connection

---

*El **Método base On Web Connection** puede llamarse en los siguientes casos:*

- *el servidor web recibe una petición que comienza por el URL 4DCGI.*
- *el servidor web recibe una petición inválida.*

*Para mayor información, consulte el párrafo "**Llamadas al Método de base On Web Connection**" abajo.*
*La petición debe haber sido aceptada previamente por el **Método base On Web Authentication** (si existe) y el servidor web debe lanzarse.*
*El **Método base On Web Connection** recibe seis parámetros de tipo texto, pasados por 4D ($1, $2, $3, $4, $5 y $6). Los contenidos de estos parámetros son los siguientes:*

Parámetros	Tipo	Descripción
$1	Texto	URL
$2	Texto	Encabezado HTTP + cuerpo HTTP (hasta 32 kb de límite)
$3	Texto	Dirección IP del navegador
$4	Texto	Dirección IP llamada del servidor
$5	Texto	Nombre de usuario
$6	Texto	Contraseña

Debe declarar estos seis parámetros de esta manera:

` Método de base On Web Connection

```
C_TEXT($1;$2;$3;$4;$5;$6)
```

` Código para el método

- ***Datos extra del URL***

  *El primer parámetro ($1) es el **URL** introducido por el usuario en el área de ubicación de su navegador web, menos la dirección local.*
  *Tomemos el ejemplo de una conexión de Intranet. Supongamos que la dirección **IP** de su equipo servidor web 4D es 123.4.567.89. La tabla siguiente muestra los valores de $1 dependiendo del URL introducido en el navegador web:*

URL introducido en el navegador	Valor del parámetro $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Clientes	/Clientes
http://123.4.567.89/Clientes	/Clientes
http://123.4.567.89/Clientes/Añadir	/Clientes/Añadir
123.4.567.89/Hacer_esto/Si_OK/Hacer_eso	/Hacer_esto/Si_OK/Hacer_eso

  Note que usted es libre de utilizar este parámetro a su conveniencia. 4D simplemente ignora los valores pasados más allá de la parte local del URL. Por ejemplo, puede establecer una convención donde el valor "/Clientes/Añadir" signifique "ir directamente a añadir un nuevo registro en la tabla [Clientes]." Suministrando a los usuarios web de su base una lista de posibles valores y/o marcadores por defecto, puede ofrecer atajos a las diferentes partes de su aplicación. De esta forma, los usuarios web pueden acceder rápidamente a los recursos de su sitio web sin tener que navegar cada vez que se conecten a su base.

***Advertencia:** para evitar que un usuario acceda directamente a una base con un marcador creado durante una sesión anterior, 4D intercepta todo URL que corresponda a uno de los URLs estándar de 4D.*

- ***Encabezado y cuerpo de la petición HTTP***
  *El segundo parámetro ($2) es el encabezado y el cuerpo de la petición HTTP enviada por el navegador web. Note que esta información se pasa a su **Método base On Web Connection** tal como está. El contenido varía en función del tipo de navegador web que esté intentando la conexión*
  Con Safari corriendo en Mac OS, puede recibir un encabezado similar a este:

```
GET /favicon.ico HTTP/1.1
Referer: http://123.45.67.89/4dcgi/test
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; fr-fr) AppleWebKit/523.10.3 (KHTML, like Gecko) Version/3.0.4
Safari/523.10
Cache-Control: max-age=0
Accept: */*
Accept-Language: fr-fr
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: 123.45.67.89
```

Con Microsoft Internet Edge en Windows, puede recibir un encabezado similar a este:

```
GET / HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg, application/x-ms-xbap,
application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Accept-Language: fr-FR
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729;
.NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C)
Accept-Encoding: gzip, deflate
Host: 123.45.67.89
Connection: Keep-Alive
```

Si su aplicación manipula esta información, es su decisión analizar el encabezado y el cuerpo.
***Nota:** por razones de rendimiento, el tamaño de estos datos no puede ser mayor a 32 KB. Si el tamaño es mayor, los datos son truncados por el servidor 4D HTTP.*

- ***Dirección IP del cliente web***
  El tercer parámetro $3 recibe la dirección IP del equipo navegador. Esta información puede permitirle distinguir entre las conexiones de Intranet e Internet.
  *Nota**:** 4D devuelve direcciones IPv4 en un formato híbrido IPv6 escrito con un prefijo de 96 bits, por ejemplo ::ffff:192.168.2.34 para la dirección IPv4 192.168.2.34. Para mayor información, consulte la sección **Soporte de IP v6**.*

- ***Dirección IP del servidor***
  *El cuarto parámetro $4 recibe la dirección IP solicitada del servidor web 4D. 4D autoriza el multi-homing, el cual le permite explotar equipos que tengan más de una dirección IP. Para mayor información, consulte la sección **Parámetros del servidor web**.*

- ***Nombre de usuario y contraseña***
  *Los parámetros $5 y $6 reciben el nombre de usuario y la contraseña introducidos por el usuario en la caja de diálogo estándar de identificación mostrada por el navegador. Esta caja de diálogo aparece para cada conexión, si la opción **Utilizar contraseñas** ha sido seleccionada en las Preferencias (ver sección **Seguridad de las conexiones**).*

***Nota:** si el nombre de usuario enviado por el navegador existe en 4D, el parámetro $6 (la contraseña del usuario) no se devuelve por razones de seguridad.*

Llamadas al Método de base On Web Connection

*El **Método base On Web Connection** puede utilizarse como punto de entrada para el servidor web 4D, bien sea utilizando el URL especial 4DCGI, o utilizando los URLs de comando personalizados.*

***Advertencia:** la llamada de un comando 4D que muestra un elemento de interfaz (**DIALOG**, **ALERT**...) termina el procesamiento del método.*

*El **Método base On Web Connection** se llama en los siguientes casos:*

- *Cuando 4D recibe el URL /4DCGI. El método base se llama con el URL /4DCGI/<action> en $1.*
- *Cuando una página web llamada con un URL de tipo <ruta>/<archivo> no se encuentra. El método de base se llama con el URL (*).*
- *Cuando una página web se llama con un URL del tipo <file>/ y ninguna página ha sido definida por defecto. El método de base se llama con el URL (*).*

(*) En estos casos particulares, el URL recibido en $1 NO comienza por el carácter "/".

Método base On Web Close Process

---

*El **Método base On Web Close Process** es llamado por el servidor web de 4D cada vez que una sesión web se va a cerrar. Una sesión puede ser cerrarse en los siguientes casos:*

- *cuando se alcanza el número máximo de sesiones simultáneas (100 por defecto, modificable utilizando el comando **WEB SET OPTION**), y 4D necesita crear nuevas (4D automáticamente destruye el proceso de la sesión inactiva más antigua),*
- *cuando se alcanza el periodo máximo de inactividad del proceso de la sesión (480 minutos por defecto, modificable vía el comando **WEB SET OPTION**),*
- *cuando se llama al comando **WEB CLOSE SESSION**.*

*Cuando se llama a este método base, el contexto de la sesión (variables y selecciones generadas por el usuario) es aún valido. Esto significa que puede guardar los datos relativos a la sesión para poder usarlos posteriormente, más específicamente utilizando **Método base On Web Connection**.*
***Nota:** en el contexto de una sesión 4D Mobile (que puede generar varios procesos), el **Método base On Web Close Process** se llama para cada proceso web cerrado, lo que permite guardar todo tipo de datos (variables, selección, etc.) generados por el proceso de sesión 4D Mobile.*

*Un ejemplo de uso del **Método base On Web Close Process** se presenta en la sección **Gestión de las sesiones web**.*

Gestión de las sesiones web

El servidor web de 4D ofrece un mecanismo simple y completo de gestión de sesiones usuario. Este mecanismo automático permite a los clientes web reutilizar el mismo contexto (selecciones e instancias de variables) de una petición a otra.

Esto se hace a través de una cookie privada creada por 4D: "4DSID". En cada petición cliente web, 4D verifica la presencia y el valor de la cookie 4DSID:

- *Si la cookie tiene un valor, 4D trata de encontrar el contexto de origen de la cookie entre los contextos existentes,*
  - *Si el contexto se encuentra, se reutiliza para llamada, el método **Compiler_Web** no se ejecuta,*
  - *Si no se encuentra contexto, 4D crea uno nuevo,*
- *Si la cookie no tiene un valor o si no se presenta (por vencimiento, por ejemplo), 4D crea un nuevo contexto.*

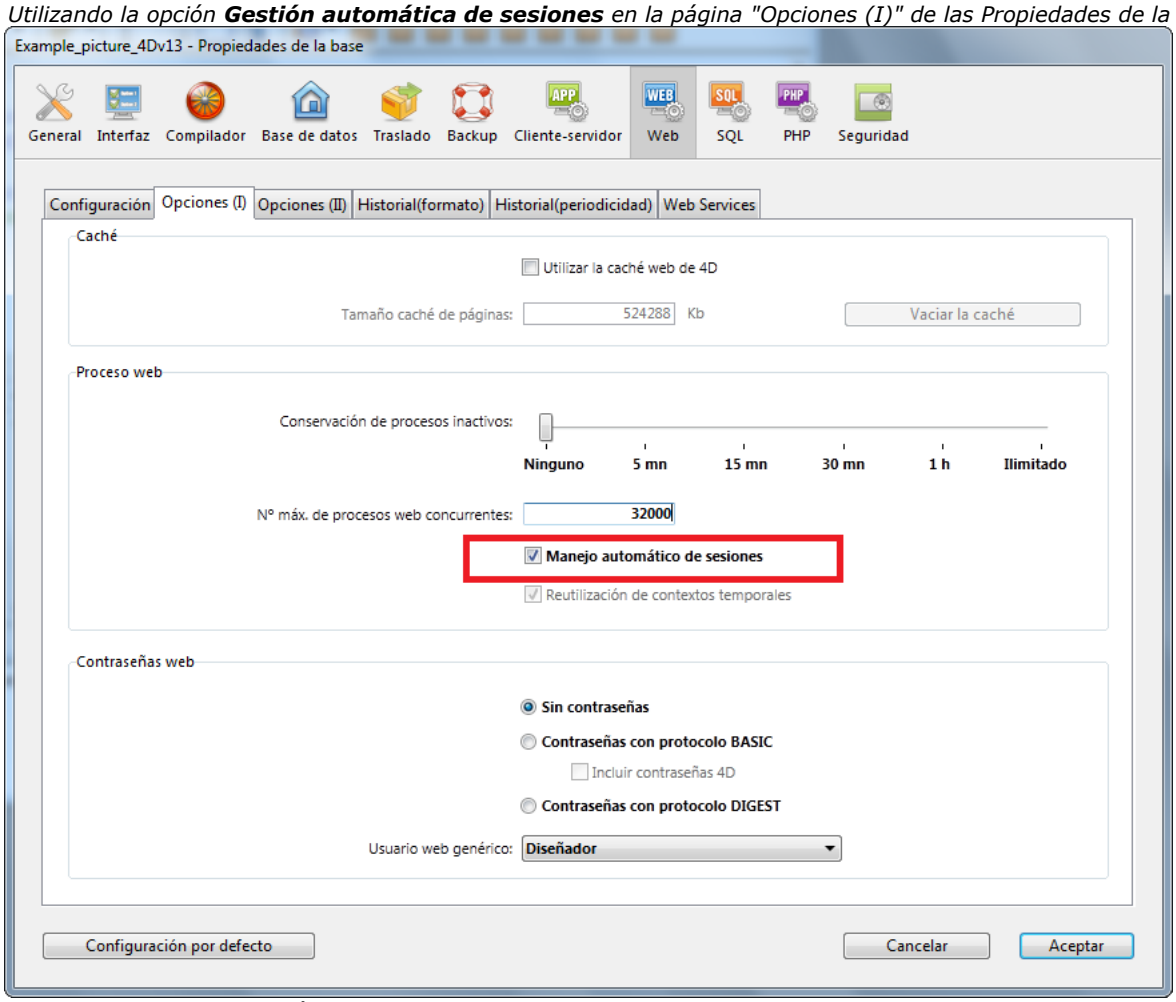
Activación y desactivación del mecanismo

El mecanismo de gestión de las sesiones debe activarse en su servidor web 4D para que pueda utilizarlo en su aplicación.

Por defecto, este mecanismo se activa en las bases de datos creadas con 4D v13 y versiones posteriores. Sin embargo, por razones de compatibilidad, se desactiva en bases de datos convertidas desde una versión anterior de 4D. Debe activarlo explícitamente para poder beneficiarse de esta nueva funcionalidad.

Hay dos formas de activar la gestión automática de sesiones:

- *Utilizando la opción **Gestión automática de sesiones** en la página "Opciones (I)" de las Propiedades de la base:*



En este caso, la configuración es permanente; se guarda en el disco.

- *Utilizando la opción Web Keep session del comando **WEB SET OPTION**. En este caso, la configuaración se aplica únicamente a la sesión y "anula" la configuración definida en la configuración de la base.*

En ambos casos, la configuración es local a la máquina; de manera que puede ser diferente en el servidor web 4D y en los servidores web 4D remotos.

Vencimiento de cookies y conservación de los contextos

*La vida útil de una cookie inactiva es de 8 horas (480 minutos) por defecto, esto se puede cambiar con el comando **WEB SET OPTION**. Puede establecer un tiempo de vida diferente para las cookies (opción Web Inactive session timeout) y para los procesos asociados a las sesiones en el servidor (opción Web Inactive process timeout): por ejemplo, es posible que desee que un "carrito de compras" permanezca válido durante 24 horas pero, con por razones de optimización, no desea mantener el proceso durante tanto tiempo. En este caso, se puede establecer una duración de vida del proceso de 4 horas, por ejemplo. Al final de este periodo, se llama el **Método base On Web Close Process** y se pueden almacenar las variables y las selecciones relacionadas con la sesión antes de matar el proceso. La próxima vez que se conecte el cliente web (hasta 24 horas después), la cookie se envía de vuelta al servidor y puede cargar la información de la sesión en el **Método base On Web Connection** (ver el ejemplo a continuación).*

*Si es necesario, puede utilizar el comando **WEB CLOSE SESSION** para forzar la expiración de la cookie, en cualquier momento y por lo tanto cerrar la sesión.*

Destrucción de contextos inactivos

*4D destruye automáticamente los contextos inactivos más antiguos cuando se alcanza el número máximo de contextos guardados (este número es de 100 por defecto y se puede cambiar con el comando **WEB SET OPTION**).*
*Cuando el contexto está a punto de ser destruido (cierre del proceso web asociado), se llama al **Método base On Web Close Process** , que le permite guardar las selecciones y las variables del contexto, a la espera de su utilización posterior.*

Ejemplo

*Este ejemplo muestra lo fácil que es administrar sesiones utilizando el **Método base On Web Connection** y el **Método base On Web Close Process** .*
*Este es el código del **Método base On Web Connection**:*

```
  // On Web Connection (or On Web Authentication)
C_TEXT(www_SessionID)
If(www_SessionID=WEB Get Current Session ID)
  // Compiler_Web no se llama
  // Todas las variables y las selecciones ya existen
    ...
Else
  // Compiler_Web se acaba de ejecutar.
  // Esta es una nueva sesión, ninguna variable o selección existe
  // Registre el ID de la nueva sesión
    www_SessionID:=WEB Get Current Session ID

  // Inicialización de la sesión
  // Definición de las selecciones
  // Recuperación del usuario seleccionado
    QUERY([Usuario];[Usuario]Login=www_Login)
    QUERY([prefs];[prefs]Login=www_Login)

  // Coordenadas del empleado
    QUERY([empleados];[empleados]Nombre=[usuario]nombre)
    QUERY([empresa];[empresa]Nombre=[usuario]empresa)

  // Definición de las variables
  // Lectura de las preferencias para este usuario
    SELECTION TO ARRAY([prefs]nombre;prefNames;[prefs]valores;prefValues)
    www_UserName:=[User]Name
    www_UserMail:=[User]mail

  // Sesión inicializada
End if
```

*Código del **Método base On Web Close Process** :*

```
  // On Web Session Suspend
  // Después de un periodo de inactividad o en caso de ser necesario, 4D cierra la sesión
C_TEXT(www_SessionID)
www_SessionID:=""
  // Se guarda la información de la sesión
  // Guardamos las preferencias del usuario conectado previamente
QUERY([prefs];[prefs]Login=www_Login)  // conservado en la sesión
ARRAY TO SELECTION(prefNames;[prefs]nombre;prefValues;[prefs]valores)

  // Importante: se mata el proceso
  // 4D borra las variables, selecciones, etc.
```

En caso de rechazo de cookies

Como el mecanismo de gestión de sesiones está basado en la utilización de cookies, el servidor HTTP de 4D no puede mantener una sesión si el cliente web no acepta las cookies. En este caso, cada petición es tratada como una nueva conexión y el método ***Compiler_Web*** *se ejecuta en cada conexión.*

Puede verificar si el soporte para cookies con el comando ***WEB GET HTTP HEADER****.*

Sesiones y gestión de las direcciones IP

El servidor HTTP de 4D registra la IP que inicia una sesión. Si un cliente web en una dirección IP diferente intenta acceder a una sesión existente, se devuelve el error HTTP 400 al cliente.

Páginas semidinamicas

---

*El servidor web de 4D le permite utilizar **páginas semi-dinámicas**.*

*Estas páginas son 'templates' HTML que contienen las **Etiquetas HTML 4D**, es decir, una mezcla de código HTML estático y referencias 4D añadidas vía las etiquetas de transformación tales com 4DHTML, 4DIF, o 4DINCLUDE. Estas etiquetas se insertan como comentarios de tipo HTML (<!--#Tag Contents-->) en el código fuente HTML.*

***Nota:** una sintaxis alternativa basada en $ se utiliza en ciertas condiciones para las etiquetas 4DHTML, 4DTEXT y 4DEVAL con el fin de hacerlas compatibles con XML. Para más información, consulte la sección **Nueva sintaxis con $ para 4DTEXT, 4DHTML, 4DEVAL**.*

Cuando estas páginas son enviadas por el servidor HTTP, se analizan y las etiquetas que contienen se ejecutan y se reemplazan con los datos resultantes. Las páginas recibidas por los navegadores son, una combinación de elementos estáticos y los valores procedentes de 4D.

Principios

*Puede utilizar programación para dar valores por defecto a los objetos HTML incluyendo <!--#4DTEXT NomVar--> en el campo **valor**del objeto HTML, donde **NomVar** es el nombre de la variable proceso 4D como se definió en el proceso web actual. Este es el nombre que usted rodea con la notación HTML estándar para los comentarios <!--#...-->.*
***Nota**: algunos editores HTML no aceptan <!--#4DTEXT NomVar--> en el campo **valor** de los objetos HTML. En este caso, deberá digitalizarlo directamente en el código HTML.*

De hecho, la sintaxis <!--#4DTEXT NomVar--> le permite insertar los datos 4D en cualquier parte de la página HTML. Por ejemplo, si escribe:

<P>Bienvenido a <!--#4DTEXT vtNomSitio-->!</P>

El valor de la variable 4D vtNombreSitio se insertará en la página HTML.

Este es un ejemplo:

```
//El siguiente código 4D asigna "4D4D" a la variable proceso vs4D vs4D:="4D4D" // Luego envía la página HTML "AnyPage.HTM" SEND HTML FILE("AnyPage.HTM")
```

*La fuente de la página HTML **AnyPage.HTM** es la siguiente:*

```
<html> <head> <title>AnyPage</title> <script language="JavaScript"><!-- function Is4DWebServer(){ return (document.frm.vs4D.value=="4D4D") </p><p>} function HandleButton(){ if(Is4DWebServer()){ alert("You are connected to 4D Web Server!") } else { alert("You are NOT connected to 4D Web Server!") } //--></script> </head> <body> <form action="/4DACTION/WWW_STD_FORM_POST" method="post" name="frm"> <p><input type="hidden" name="vs4D" value="<!--#4DTEXT vs4D-->"</p> <p><a href="JavaScript:HandleButton()"><img src="AnyGIF.GIF" border=0 align=bottom></a></p> <p><input type="submit" name="bOK" value="OK"></p> </form> </body> </html>
```

*La etiqueta <!--#4DTEXT --> permite igualmente insertar las **expresiones 4D** en las páginas enviadas (campos, elementos de arrays, etc.). La operación de esta etiqueta con este tipo de datos es idéntica al de las variables. También puede insertar código HTML en las variables 4D con la ayuda de la etiqueta 4DHTML. Otras etiquetas tales como 4DIF permiten controlar el código ejecutado. El conjunto de etiquetas ejecutables se describe en la sección **Etiquetas HTML 4D**.*

Procesamiento de las etiquetas

*El análisis del contenido de las páginas semidinámicas enviadas por 4D se efectúa en el momento de la llamada a los comandos **WEB SEND FILE** (.htm, .html, .shtm, .shtml) o **WEB SEND BLOB** (blob de tipo texto/html) o **WEB SEND TEXT**, como también cuando se envían páginas llamadas utilizando URLs. En este último caso, por razones de optimización las páginas que tienen el sufijo ".htm" y "html" NO se analizan.*
*Para forzar el análisis de páginas HTML en este caso, debe añadir el sufijo ".shtm" o ".shtml" (por ejemplo, http://www.server.com/dir/page.shtm). Un ejemplo de uso de este tipo de página se presenta en la descripción del comando **WEB GET STATISTICS**.*
Las páginas XML (.xml, .xsl) y las páginas WML ((.wml) también son tenidas en cuenta y analizadas por 4D (ver sección [#title id="778"/]).

*El análisis también puede efectuarse fuera del contexto web cuando utiliza el comando **PROCESS 4D TAGS**.*

Internamente, el analizador trabaja con las cadenas UTF-16, pero los datos a analizar pueden haber sido codificados de manera diferente. Cuando las etiquetas contienen texto (por ejemplo, 4DHTML), 4D convierte los datos cuando es necesario dependiendo de su origen y de la información disponible (charset). A continuación están los casos donde 4D analiza las etiquetas contenidas en las páginas HTML, como también las conversiones efectuadas:

Acción	Análisis del contenido de las páginas enviadas	Soporte de sintaxis $ (*)	Conjunto de caracteres utilizado para el análisis de las etiquetas
Páginas llamadas vía URLs	X, excepto páginas con extensiones ".htm" o ".html"	X, excepto páginas con sufijos ".htm" o ".html"	Uso del charset pasado como parámetro del encabezado "Content-Type" de la página. Si no hay, busca de una etiqueta META-HTTP EQUIV con un charset. De lo contrario, utiliza el conjunto de caracteres por defecto del servidor HTTP
Comando **WEB SEND FILE**	X	-	Uso del charset pasado como parámetro del encabezado "Content-Type" de la página. Si no hay, busca una etiqueta META-HTTP EQUIV con un charset. De lo contrario, utilice el conjunto de caracteres por defecto para el servidor HTTP
Comando **WEB SEND TEXT**	X	-	No es necesaria la conversión
Comando **WEB SEND BLOB**	X, si el BLOB es de tipo "text/html"	-	Uso del charset definido en el encabezado "Content-Type" de la respuesta. De lo contrario, utiliza el conjunto de caracteres por defecto del servidor HTTP
Inclusión por la etiqueta <!--#4DINCLUDE-->	X	X	Uso del charset pasado como parámetro del encabezado "Content-Type" de la página. Si no hay, busca una etiqueta META-HTTP EQUIV con un charset. De lo contrario, utilice el conjunto de caracteres por defecto para el servidor HTTP
Comando **PROCESS 4D TAGS**	X	X	Datos texto: no conversión. Datos BLOB: conversión automática del conjunto de caracteres Mac-Roman por compatibilidad

(*) La sintaxis alternativa utiliza el $ disponible para las etiquetas 4DHTML, 4DTEXT y 4DEVAL (ver la sección).

Encapsulación de JavaScript

4D soporta código fuente JavaScript encapsulado en los documentos HTML y también la inserción de archivos JavaScript .js en los documentos HTML (por ejemplo <SCRIPT SRC="...").

*Utilizando **WEB SEND FILE** o **WEB SEND BLOB**, usted envía una página que haya preparado en un editor HTML o creada por programación utilizando 4D y guardada como documento en disco. En ambos caso, tiene control total de la página. Puede insertar scripts JavaScript en la sección HEAD del documento y utilizar los scripts con una etiqueta FORM. En el ejemplo anterior, el script reenvía el formulario "frm" porque usted pudo darle nombre al formulario. Puede igualmente activar, aceptar o rechazar el envío del formulario a nivel de la etiqueta FORM.*
***Nota**: 4D soporta el transporte de Applets Java.*

URLs y acciones de formularios

El servidor web 4D ofrece diferentes URLs y acciones de formularios HTML especiales que permiten implementar diferentes acciones en su base.
Estos URLs son los siguientes:

- *4DACTION/, para asociar un objeto HTML a un método de proyecto de su base,*
- *4DCGI/, permite llamar el **Método base On Web Connection** desde todo objeto HTML.*
- *4DSYNC/, para sincronizar los datos de las tablas.*

Adicionalmente, el servidor web 4D acepta varios URLs adicionales:

- */4DSTATS*, */4DHTMLSTATS*, */4DCACHECLEAR* y */4DWEBTEST*, *para permitirle obtener información sobre el funcionamiento de su sitio Web 4D. Estos URLs se describen en la sección **Información sobre el sitio web**.*
- */4DWSDL*, *permite el acceso al archivo de declaración de los Servicios web publicados en el servidor. Para mayor información, consulte la sección **Web Services (Servidor)** y el Manual de Diseño.*

URL 4DACTION/

Sintaxis: *4DACTION/MiMetodo{/Param}*

Uso: *URL o acción de formulario.*

*Este URL permite asociar un objeto HTML (texto, imagen, botón...) a un método de proyecto 4D. Este enlace será del tipo /4DACTION/MiMetodo/Param donde **MiMetodo** es el nombre del método de proyecto 4D a ejecutar cuando el usuario hace clic en el enlace y Param un parámetro opcional de tipo Texto pasado al método en $1 (ver el párrafo "Los parámetros Texto pasados a los métodos vía los URLs").*

*Cuando 4D recibe una petición /4DACTION/MiMetodo/Param, se llama al **Método de base On Web Authentication** (si existe). Si devuelve **True**, se ejecuta el método **MiMetodo**.*

4DACTION/ puede estar asociado a un URL en una página web estática. La sintaxis del URL debe ser de esta forma:

```
<A HREF="/4DACTION/MyMethod/Param"> Do Something</A>
```

*El método de proyecto **MiMetodo** generalmente debe devolver una "respuesta" (envío de página HTML utilizando **WEB SEND FILE** o **WEB SEND BLOB**, etc.). Asegúrese de hacer los procesos tan cortos como sea posible para no bloquear el navegador.*
***Nota**: un método llamado por 4DACTION no debe llamar a los elementos de interfaz (**DIALOG**, **ALERT**...).*
***Advertencia**: para que un método 4D pueda ejecutarse vía el URL 4DACTION/, debe tener el atributo "Disponible vía 4DACTION, 4DMETHOD y 4DSCRIPT" (deseleccionado por defecto), definido en las propiedades del método. Para mayor información sobre este punto, consulte la sección **Seguridad de las conexiones**.*

Ejemplo 1

Este ejemplo describe la asociación del URL 4DACTION/ con un objeto HTML imagen con el fin de mostrar dinámicamente una imagen en la página. Inserte la siguiente instrucción en una página HTML estática:

```
<IMG SRC="/4DACTION/PICTFROMLIB/1000">
```

*El método **PICTFROMLIB** es el siguiente:*

```
C_TEXT($1) // Este parámetro debe declararse siempre
C_PICTURE($PictVar)
C_BLOB($BlobVar)
C_LONGINT($Number)
 // Recuperamos el número de imagen en la cadena $1
$Number:=Num(Substring($1;2;99))
GET PICTURE FROM LIBRARY($Number;$PictVar)
PICTURE TO BLOB($PictVar;$BlobVar;".gif")
WEB SEND BLOB($BlobVar;"Pict/gif")
```

4DACTION para enviar formularios

El servidor web 4D permite utilizar formularios "enviados", es decir las páginas HTML estáticas que envían datos al servidor web y recuperar fácilmente el conjunto de los valores. La acción del formulario debe obligatoriamente comenzar por /4DACTION/NomMetodo.
***Nota**: un formulario puede ser enviado por medio de dos métodos (ambos pueden utilizarse con 4D):*

- *POST, generalmente utilizado para añadir datos al servidor web, en una base de datos,*
- *GET, generalmente utilizado para hacer peticiones al servidor web, datos que vienen de una base.*

*En este caso, cuando el servidor web recibe un formulario enviado, llama al **Método de base On Web Authentication** (si existe). Si devuelve **True**, se ejecuta el método **NomMetodo**. En este método, debe llamar al comando **WEB GET VARIABLES** con el fin de recuperar el nombre y el valor de todos los campos incluidos en una página HTML enviada al servidor.*

***Nota de compatibilidad**: en bases convertidas, si la opción "Asignación de variables automática" en la **Página Compatibilidad** está seleccionada, el método proyecto especial COMPILER_WEB (si existe) se llama en primero; 4D recupera los valores de los campos HTML que se encuentran en el formulario y llena automáticamente las variables 4D en el método llamado con sus contenidos si tienen el mismo nombre. Este funcionamiento es obsoleto. Para más información, consulte la sección **Asociar objetos 4D a objetos HTML** .*
La sintaxis HTML a aplicar en el formulario es del siguiente tipo:

- *Para la definición de la acción del formulario:*

```
<FORM ACTION="/4DACTION/NomMetodo" METHOD=POST>
```

- *Para definir un campo en un formulario:*

```
<INPUT TYPE=Tipodecampo NAME=Nombredelcampo VALUE="Valorpordefecto">
```

Para cada campo en el formulario, 4D define el valor del campo para el valor de la variable con el mismo nombre.

Ejemplo 2

En una base web 4D, queremos que los navegadores puedan buscar registros utilizando una página estática HTML. Esta página se llama "buscar.html". La base contiene otras páginas estáticas que le permiten, por ejemplo, mostrar el resultado de la búsqueda ("resultados.html"). El tipo POST ha estado asociado a la página, como también la acción /4DACTION/SEARCH.
Este es el código HTML que corresponde a esta página:

```
#codeHTML]<FORM ACTION="/4DACTION/PROCESSFORM" METHOD=POST>
<INPUT TYPE=TEXT NAME=VNAME VALUE=""><BR>
<INPUT TYPE=CHECKBOX NAME=EXACT VALUE="Word">Whole word<BR>
<INPUT TYPE=SUBMIT NAME=OK VALUE="Search">
</FORM>[#/codeHTML]
```

*Durante la entrada de datos, escriba "ABCD" en el área de entrada, seleccione la opción "Palabra completa" y valide haciendo clic en el botón **Buscar**.*
En la petición enviada al servidor web:

```
VNAME="ABCD"
vEXACT="Word"
OK="Search"
```

*4D llama al **Método de base On Web Authentication** (si existe), luego se llama al método proyecto PROCESARFORM, que es el siguiente:*

```
C_TEXT($1) //obligatorio para modo compilado
C_LONGINT($vName)
C_TEXT(vNAME;vLIST)
ARRAY TEXT($arrNames;0)
ARRAY TEXT($arrVals;0)
WEB GET VARIABLES($arrNames;$arrVals) //recuperamos todas las variables del formulario
$vName:=Find in array($arrNames;"vNAME")
vNAME:=$arrVals{$vName}
If(Find in array($arrNames;"vEXACT")=-1)  //Si la opción no ha sido seleccionada
   vNAME:=VNAME+"@"
End if
QUERY([Jockeys];[Jockeys]Name=vNAME)
FIRST RECORD([Jockeys])
While(Not(End selection([Jockeys])))
   vLIST:=vLIST+[Jockeys]Name+" "+[Jockeys]Tel+"<BR>"
   NEXT RECORD([Jockeys])
End while
WEB SEND FILE("results.htm") //Envío de la lista en el formulario results.htm
 //que contiene una referencia a la variable vLIST
 //por ejemplo <!--4DHTML vLIST-->
 //...
End if
```

URL 4DCGI/

Sintaxis: *4DCGI/<action>*

Uso: *URL.*

Cuando el servidor web 4D recibe el URL /4DCGI/<action> se llama el ***Método de base On Web Authentication*** *(si existe). Si devuelve* ***True****, el servidor web llama al* ***Método base On Web Connection*** *enviando el URL "tal cual" a $1.*
El URL 4DCGI/ URL no corresponde a ningún archivo. Su papel es llamar 4D utilizando el ***Método base On Web Connection****. El parámetro "<action>" puede contener todo tipo de información.*
Este URL le permite efectuar todo tipo de acción. Sólo necesita probar el valor de $1 en el ***Método base On Web Connection*** *o en uno de sus submétodos y realizar en 4D la acción apropiada. Por ejemplo, puede crear páginas HTML estáticas totalmente personalizadas para añadir, buscar, u ordenar registros, o generar imágenes GIF rápidamente. Ejemplos de cómo utilizar este URL se encuentran en las descripciones de los comandos* ***PICTURE TO BLOB*** *y* ***WEB SEND HTTP REDIRECT****.*
*Después de una acción, debe devolverse una "respuesta", utilizando uno de los comandos de envío de datos (****WEB SEND FILE****,* ***WEB SEND BLOB****, etc.).*

Advertencia*: asegúrese de ejecutar las acciones más cortas posibles, con el fin de no bloquear el navegador.*

Los parámetros de texto pasados a los métodos 4D llamados vía los URLs

4D envía los parámetros de Texto a los métodos 4D llamados por los URLs especiales (4DACTION/ y 4DCGI/). He aquí algunas observaciones sobre estos parámetros:

- *Aunque no utilice estos parámetros, debe declararlos explícitamente con el comando* ***C_TEXT****, de lo contrario se producirán errores runtime cuando acceda por web a una base ejecutada en modo compilado. El mensaje es del tipo*
  *"****Error en código dinámico***
  Parámetros incorrectos en un comando EXECUTE
  Nombre del método:
  Número de línea:
  Descripción: [<fecha y hora>]"
  Este error runtime se produce por el hecho de que el parámetro texto $1 no fue declarado en el método 4D llamado al hacer clic en el enlace HTML. Como el contexto de la ejecución es la página HTML actual, el error no hace referencia específicamente a la línea del método. Declarar explícitamente el parámetro texto $1 permite eliminar estos errores:

```
//Método proyecto M_SEND_PAGE
C_TEXT($1) // Este parámetro debe declararse explícitamente
//...
WEB SEND FILE($mypage)
```

- *El parámetro $1 devuelve los datos adicionales ubicados al final del URL y pueden utilizarse para pasar los datos del entorno HTML al entorno 4D.*

Parámetros para declarar explícitamente en el método 4D llamado

Debe declarar diferentes parámetros en función del origen y de la naturaleza de la llamada del método 4D.

- ***Método de base On Web Authentication*** *(si existe) y* ***Método base On Web Connection***
  Debe declarar los seis parámetros de la conexión:

```
` Método base On Web Connection
C_TEXT($1;$2;$3;$4;$5;$6) `Estos parámetros deben declararse explícitamente
```

- *Método llamado por el URL 4DACTION/*
  Debe declarar el parámetro $1:

```
` Método llamado por el URL 4DACTION/
C_TEXT($1)
```

- *Método llamado por etiqueta 4DSCRIPT/ como un comentario HTML en un documento*
  El método debe devolver un valor en $0. Debe declarar el parámetro $0 y $1:

```
//Método llamado por la etiqueta 4DSCRIPT/ como un comentario HTML
C_TEXT($0;$1) //Estos parámetros DEBEN ser declarados explícitamente
```

URL 4DSYNC/

Sintaxis:

4DSYNC/$catalog{/NomTabla}
4DSYNC/NomTabla{/NomTabla}{/NomCampo1,...,NomCampoN}{Params}

Uso: *URL en método POST o GET*

Este URL sincroniza los datos de las tablas de la base 4D local con una base remota vía HTTP. Se utiliza para sincronizar una base 4D con una aplicación cliente instalada en un Smartphone o con toda aplicación HTTP de terceros.

El URL 4DSYNC/ se utiliza en método GET para recuperar los datos de la base 4D o en método POST para actualizar los datos en la base 4D.

Estas son las diferentes peticiones HTTP utilizables:

- *GET /4DSYNC/$catalog*
  Devuelve la lista de las tablas de la base y el número de registros que contiene.Por ejemplo para una estructura con dos tablas PERSONS (2 registros) e INVOICES (3 registros), la sintaxis: http://localhost/4DSYNC/$catalog/ devuelve en el navegador : PERSONS 2 INVOICES 3

- *GET /4DSYNC/$catalog/NomTabla*
  Devuelve la descripción de la estructura de NomTabla (formato XML).

- *GET /4DSYNC/NomTabla/NomCampo1{,NomCampo2},...*
  Devuelve los datos de los campos NomCampo de la tabla NomTabla.

- *GET /4DSYNC/TableName/FieldName1?$stamp=0&$format=json*
  Devuelve los datos del campo NomCampo en la tabla NomTabla a partir del stamp 0 y al formato json.

- *POST /4DSYNC/NomTabla/NomCampo1{,NomCampo2},...*
  Integra en la base 4D las modificaciones efectuadas en el cliente.

***Nota:** el formato utilizado para el intercambio de datos es el JSON (JavaScript Objet Notation). La gramática completa está disponible desde el servicio de soporte técnico de 4D.*

***Nota:** para que los mecanismos de sincronización se activen, la opción **Autorizar el acceso vía los URLs 4DSYNC** debe ser seleccionada en la página "Web/Configuración" de las Propiedades de la base (ver a continuación). De lo contrario, las peticiones que contienen el URL 4DSYNC fallarán.*

Notas sobre sincronización con HTTP

Cuando utilice el URL 4DSYNC/ debe tener en cuenta los siguientes principios:

- *4D trabaja con la estructura virtual de la base, si existe. En este caso, los parámetros NomTabla y NomCampo corresponden a los nombres de tablas y campos que se han especificado utilizando los comandos **SET FIELD TITLES** y **SET TABLE TITLES**. Note que el alcance de estos comandos es la sesión y al utilizar 4D Server debe llamarlos en procedimiento almacenado en el servidor.*
- *La replicación de los campos de tipo Blob e Imagen no es soportada.*
- *Para poder sincronizar los datos:*
  - *El servidor HTTP debe lanzarse.*
  - *La opción **Autorizar el acceso a la base de datos vía los URLs 4DSYNC** debe estar seleccionada en la página "Web/Configuración" de las Propiedades de la base (ver la sección **Seguridad de las conexiones**).*
  - *La propiedad "Activar replicación" debe estar seleccionada para cada tabla donde quiera sincronizar los datos. Si se ha definido una estructura, estas tablas deben incluirse en esta estructura virtual. Advertencia: seleccionar esta opción publica información adicional; debe asegurarse de que el acceso a su base esté protegido (ver la descripción de esta opción en **Seguridad de las conexiones**).*
- *Cuando 4D recibe una petición /4DSYNC, el **Método de base On Web Authentication** se llama (excepto cuando la contraseña es incorrecta, ver el diagrama de conexión en la sección **Seguridad de las conexiones**). Si devuelve **True**, la petición se ejecuta, de lo contrario se rechaza.*

Asociar objetos 4D a objetos HTML

El servidor Web de 4D permite recuperar los datos "enviados", es decir, los datos introducidos por los usuarios a través de los formularios web y se envían al servidor vía los botones o los elementos de interfaz, en modo POST o GET.

*El servidor web acepta varias URLs específicas que se pueden asociar con los botones de modo que el envío del formulario desencadene el procesamiento del lado del servidor. Estas URLs se describen en la sección **URLs y acciones de formularios**.*

Este capítulo trata los principios implementados con el fin de recuperar los datos que se encuentran en los formularios que fueron devueltos al servidor.

Recibir los valores dinámicos

*Cuando el servidor web recibe un formulario "posted", 4D puede recuperar los valores de todos los objetos HTML que contiene. Este principio puede implementarse en el caso de un formulario web, enviado por ejemplo con **WEB SEND FILE** o **WEB SEND BLOB**, donde el usuario introduce o modifica valores, luego hace clic en el botón de validación. En este caso, hay dos formas de que 4D pueda recuperar los valores de los objetos HTML presentes en la petición:*

- *utilizando el comando **WEB GET VARIABLES** o*
- *utilizando los comandos **WEB GET BODY PART** y **WEB Get body part count**.*

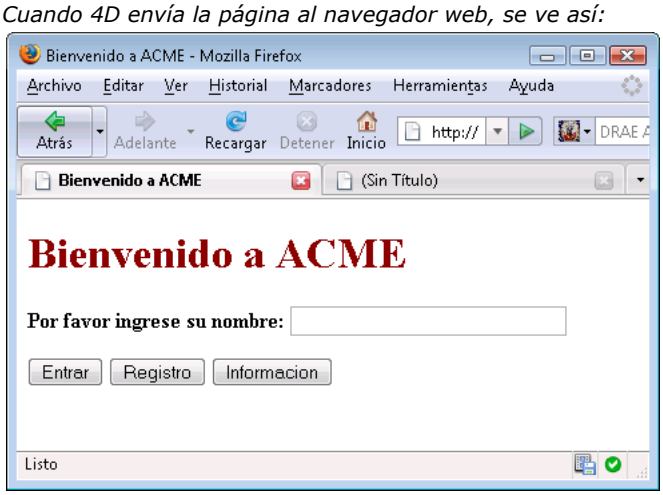
*El comando **WEB GET VARIABLES** recupera los valores en forma de texto, mientras que los comandos **WEB GET BODY PART** y **WEB Get body part count** pueden recuperar los archivos publicados, utilizando BLOBs.*

***Nota de compatibilidad (4D v13.4):** en las versiones anteriores, 4D copiaba directamente los valores de las variables recibidas vía un formulario web publicado o un URL GET, en las variables proceso 4D (en modo compilado, estas variables tenían que haber sido declaradas en la método **COMPILER_WEB**). Este funcionamiento se elimina a partir de 4D v13.4, se mantiene por compatibilidad de las bases de datos convertidas, pero puede desactivarse mediante la opción de compatibilidad **Asignación automática de variables** en la **Página Compatibilidad** de las Propiedades de la base (se recomienda desactivar esta opción y utilizar los comandos **WEB GET VARIABLES** o **WEB GET BODY PART** en sus bases).*

Considere esta página de código HTML:

```
<html> <head>  <title>Welcome</title>  <script language="JavaScript"><!-- function GetBrowserInformation(formObj){ formObj.vtNav_appName.value = navigator.appName formObj.vtNav_appVersion.value = navigator.appVersion formObj.vtNav_appCodeName.value = navigator.appCodeName formObj.vtNav_userAgent.value = navigator.userAgent return true } function LogOn(formObj){ if(formObj.vtUserName.value!=""){ return true } else { alert("Introduzca su nombre, luego intente de nuevo.") return false } } //--></script> </head> <body> <form action="/4DACTION/WWW_STD_FORM_POST" method="post"  name="frmWelcome"  onsubmit="return GetBrowserInformation(frmWelcome)">  <h1>Welcome to Spiders United</h1>  <p><b>Por favor introduzca su nombre: </b>  <input name="vtUserName" value="" size="30" type="text"></p>  <p> <input name="vsbLogOn" value="Log On" onclick="return LogOn(frmWelcome)" type="submit"> <input name="vsbRegister" value="Register" type="submit"> <input name="vsbInformation" value="Information" type="submit"></p> <p> <input name="vtNav_appName" value="" type="hidden"> <input name="vtNav_appVersion" value="" type="hidden"> <input name="vtNav_appCodeName" value="" type="hidden"> <input name="vtNav_userAgent" value="" type="hidden"></p> </form> </body> </html>
```

Cuando 4D envía la página al navegador web, se ve así:



Las principales características de esta página son:

- *Incluye tres botones Enviar: vsbEntrar, vsbRegistrar y vsbInformacion.*
- *Cuando hace clic en Log On, el envío del formulario primero es procesado por la función JavaScript **LogOn**. Si no se introduce ningún nombre, el formulario no se envía a 4D y se muestra una alerta JavaScript.*
- *El formulario tiene un método POST 4D y un script Submit (GetBrowserInformation) que copia las propiedades del navegador en cuatro objetos ocultos cuyos nombres comienzan por vtNav_App.*
- *También incluye el objeto vtNombreUsuario.*

*Examinemos ahora el método 4D (llamado **WWW_STD_FORM_POST**) que se llama cuando el usuario hace clic en uno de los botones del formulario HTML:*

```
// Recuperación del valor de las variables <p>ARRAY TEXT($arrNames;0)
ARRAY TEXT($arrValues;0)
WEB GET VARIABLES($arrNames;$arrValues)
```

```
C_TEXT($user)

Case of

 // Se hizo clic en el botón Log On
   :(Find in array($arrNames;"vsbLogOn")#-1)
      $user :=Find in array($arrNames;"vtUserName")
      QUERY([WWW Users];[WWW Users]UserName=$arrValues{$user})
      $0:=(Records in selection([WWW Users])>0)
      If($0)
         WWW POST EVENT("Log On";WWW Log information)
 // The WWW POST EVENT method saves the information in a database table
      Else

         $0:=WWW Register
 // El método WWW Register permite registrarse a un nuevo usuario web
      End if

 // Se hizo clic en el botón Register
   :(Find in array($arrNames;"vsbRegister")#-1)
      $0:=WWW Register

 // Se hizo clic en el botón Information
   :(Find in array($arrNames;"vsbInformation")#-1)
      WEB SEND FILE("userinfos.html")
End case
```

Las características de este método son:

- *Los valores de las variables 4D vtNav_appName, vtNav_appVersion, vtNav_appCodeName y vtNav_userAgent (asociadas a los objetos HTML del mismo nombre) son recuperados por el comando **WEB GET VARIABLES** a partir de los objetos HTML creados por el script JavaScript GetBrowserInformation.*
- *Fuera de las variables vsbLogOn, vsbRegister y vsbInformation relacionadas con los los tres botones Submit, sólo la que corresponde al botón en el que se hizo clic será recuperada por el comando **WEB GET VARIABLES**. Cuando el envío se efectúa por uno de estos botones, el navegador devuelve a 4D el valor del botón sobre el cual se hizo clic. Este principio le indica en cual botón se hizo clic. Note que los botones 4D (en un formulario 4D) son variables numéricas. Sin embargo, con HTML, todos los objetos son objetos texto.*

Si utiliza un objeto SELECT, este es el valor del elemento seleccionado en el objeto que es devuelto en el comando [#cmd id="683"/], y no posición del elemento en el array como en 4D.

***WEB GET VARIABLES** siempre devuelve valores de tipo texto.*

Soporte de codificación de transferencia fragmentada

A partir de 4D v15 R3, el 4D Web Server integrado soporta los archivos cargados en la codificación de transferencia fragmentada desde cualquier cliente Web. La codificación de transferencia fragmentada es un mecanismo de transferencia de datos especificado en HTTP/1.1. Permite que los datos sean transferidos en una serie de "trozos" (partes) sin conocer el tamaño de los datos finales.

***Nota:** el servidor web 4D también soporta la codificación de transferencia fragmentada desde el servidor a los clientes Web (ver **WEB SEND RAW DATA**).*

Para más información acerca de la implementación del lado del cliente para transferencias fragmentadas, consulte RFC7230 o la página correspondiente en Wikipedia.

Método de proyecto COMPILER_WEB

*El método **COMPILER_WEB**, si existe, se llama de forma sistemática cuando el servidor HTTP recibe una solicitud dinámica y llama al motor de 4D. Este es el caso, por ejemplo, cuando el servidor Web 4D recibe un formulario publicado o una URL que contiene la acción 4DCGI/. Este método está destinado a contener las directivas de digitación y/o inicialización de variables utilizadas durante los intercambios web. Es utilizado por el compilador al compilar la base. El método **COMPILER_WEB** es común para todos los formularios web. Por defecto, no existe el método **COMPILER_WEB**. Debe crearlo explícitamente.*

***Servicios web:** el método de proyecto **COMPILER_WEB** es llamado, si existe, por cada petición SOAP aceptada. Debe utilizar este método para declarar todas las variables 4D asociadas con los argumentos SOAP entrantes y para todos los métodos publicados como Servicios web. De hecho, el uso de variables proceso en los métodos de servicios web necesita que sean declaradas antes de llamar al método. Para mayor información sobre este punto, consulte la descripción del comando **SOAP DECLARATION**.*

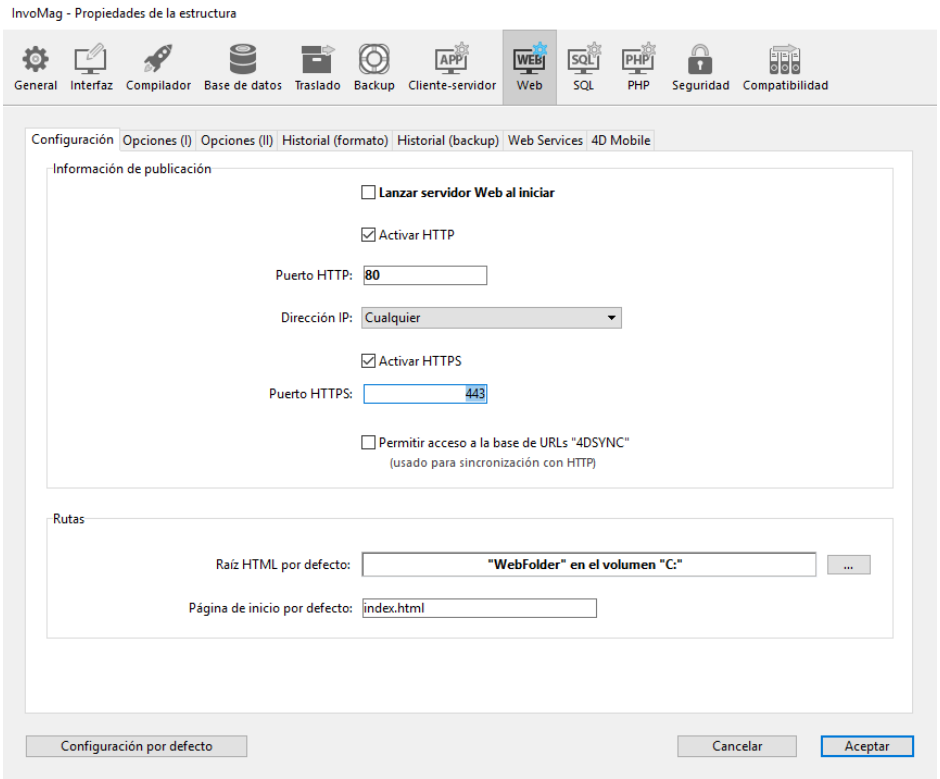
Parámetros del servidor web

*Puede configurar el funcionamiento del servidor web 4D utilizando los parámetros definidos en la página **Web** de las Propiedades de la base. Esta sección describe los parámetros de las pestañas **Configuración**, **Opciones (I)** y **(II)** de esta página.*

- *Los parámetros de las páginas **Log** se tratan en la sección **Información sobre el sitio web**.*
- *Los parámetros de la página **Web Services** se tratan en el manual de Diseño.*

***Nota de compatibilidad**: ciertos mecanismos web presentes en versiones anteriores de 4D hoy en día se consideran obsoletos (por ejemplo, "Eliminar el "/" en los URLs desconocidos). Por compatibilidad, estos mecanismos aún pueden utilizarse en bases de datos convertidas. En este caso, puede mostrarlas y si es necesario desactivarlas en la página **Compatibilidad** de las propiedades de la base.*

Página Configuración



Lanzar el servidor web al inicio

*Indica si el servidor web debe iniciarse al lanzar la aplicación 4D. Esta opción se describe en la sección **Configuración del servidor web y gestión de conexiones**.*

Activar HTTP

*Indica si el servidor Web aceptará o no conexiones no seguras. Esta opción se describe en la sección **Control del modo de conexión** de la página **Utilizar el protocolo TLS (HTTPS)**.*

Puerto TCP

*Por defecto, 4D publica una base Web en el puerto HTTP Web regular (puerto TCP), que es el puerto 80. Si ese puerto ya está siendo utilizado por otro servicio Web, debe cambiar el puerto TCP HTTP utilizado por 4D para esta base. La modificación del puerto HTTP le permite iniciar el servidor Web 4D bajo macOS sin ser el usuario raíz de la máquina (ver la sección **Configuración del servidor web y gestión de conexiones**).*

*Para hacer esto, vaya al área de entrada **Puerto HTTP** e indique un valor apropiado (un puerto TCP que no esté siendo utilizado por otro servicio TCP/IP en la misma máquina).*

***Nota:** si pasa 0, 4D utilizará el número de puerto TCP por defecto, es decir 80.*

A nivel del navegador web, debe incluir ese número de puerto HTTP no por defecto en la dirección que introduzca para conectarse a la base Web. La dirección debe tener un sufijo que consiste en dos puntos seguidos por el número de puerto. Por ejemplo, si utiliza el puerto TCP 8080, debe especificar en el navegador "123.4.567.89:8080".

***Advertencia:** si utiliza los números de puerto HTTP diferentes de los números por defecto (80 para el modo estándar y 443 para el modo TLS), tenga cuidado de no utilizar números de puerto que sean números por defecto para otros servicios que pueda utilizar simultáneamente. Por ejemplo, si piensa utilizar también el protocolo FTP en su equipo servidor Web, no utilice los puertos TCP 20 y 21, los cuales son los puerto por defecto para ese protocolo. Para conocer las asignaciones estándar de números de puerto TCP, consulte la sección **Anexo B: Números de puertos TCP** en la documentación de 4D Internet Commands. Los números de puerto inferiores a 256 están reservados para los servicios estándar y los números 256 a 1 024*

están reservados para los servicios específicos originados en las plataformas UNIX. Para máxima seguridad, especifique un número de puerto superior a estos intervalos, por ejemplo entre 2 000 ó 3 000.

Dirección IP

Puede definir la dirección IP en la cual el servidor web debe recibir las peticiones HTTP.

***Nota:** a partir de 4D v14, el servidor HTTP soporta la notación de direcciones IPv6 cuando la opción **Todos** está seleccionada en la lista "Dirección IP". Para mayor información, consulte **Soporte de IP v6**.*

*Por defecto, el servidor responde a todas las direcciones IP (opción **Todos**).*

La lista desplegable "Dirección IP" lista automáticamente todas las direcciones IP disponibles en el equipo. Cuando selecciona una dirección específica, el servidor sólo responde a las peticiones enviadas a esta dirección.

Esta funcionalidad está destinada a los servidores web 4D ubicados en equipos con múltiples direcciones TCP/IP. Por ejemplo, el caso de la mayoría de los proveedores de alojamiento web (MultiHoming). La implementación de un sistema MultiHoming requiere configuraciones específicas en el equipo servidor web:

- ***Configuración MultiHoming en Mac OS***

Para configurar un sistema MultiHoming bajo Mac OS:

1. *Abra el panel de control **TCP/IP**.*
2. *Seleccione la opción **Manual** del menú **Configuración**.*
3. *Cree un archivo de texto llamado "Dirección IP secundaria" y guárdelo en la subcarpeta Preferencias de su carpeta Sistema. Cada línea del archivo "Dirección IP secundaria" debe contener una dirección IP secundaria y si es necesario, una máscara de subred y una dirección de router para la dirección IP secundaria.*

Para mayor información consulte la documentación Apple.

- ***Configuración MultiHoming bajo Windows***
  Para configurar un sistema MultiHoming bajo Windows:

1. *Seleccione las siguientes secuencias de comandos (o sus equivalentes en función de su versión de Windows):*

   *Menú **Inicio > Panel de control** > **Conexiones de red e Internet**> **Conexión de área local** (Propiedades) > **Protocolo de Internet (TCP/IP)** > botón **Propiedades** > botón **Opciones avanzadas...** Se muestra la caja de diálogo de configuración "Parámetros avanzados TCP/IP).*
2. *Haga cli**c en el botón Agregar...**. en el área "Direcciones IP" y añada las direcciones IP adicionales.*

Puede definir hasta 5 direcciones IP diferentes. Para esta operación, puede necesitar de la ayuda de un administrador de redes.

Activar HTTPS

*Indica si el servidor web debe aceptar o no las conexiones seguras. Esta opción se describe en la sección **Control del modo de conexión** de la página **Utilizar el protocolo TLS (HTTPS)**.*

Número de puerto HTTPS

Permite modificar el número de puerto TCP/IP utilizado por el servidor web para las conexiones HTTP seguras sobre SSL (protocolo HTTPS). Por defecto, el número de puerto HTTPS es 443 (valor estándar).

Puede considerar modificar este número de puerto por dos razones principales:

- *por razones de seguridad, los ataques de piratas contra los servidores web se concentran generalmente en los puertos TCP estándar (80 y 443).*
- *bajo Mac OS X, para permitir a los usuarios “estándar” lanzar el servidor web en modo seguro, bajo Mac OS X, el uso de puertos TCP/IP reservados a la publicación web (0 a 1023) requiere de privilegios de acceso específicos: sólo el usuario "root" puede lanzar una aplicación utilizando estos puertos. Para que los usuarios estándar puedan lanzar el servidor web, una solución es modificar el número de puerto TCP/IP (ver la sección **Configuración del servidor web y gestión de conexiones**).*

Puede pasar todo valor válido (para evitar restricciones de acceso bajo Mac OS X, debe pasar un valor superior a 1023). Para mayor información sobre los números de puerto TCP, consulte el párrafo "Número de puerto TCP".

Autorizar el acceso a la base de datos vía los URLs 4DSYNC

*Esta opción controla el soporte de peticiones de sincronización HTTP que contienen los URLs /4DSYNC. Esta opción se cubre en la sección **Seguridad de las conexiones**.*

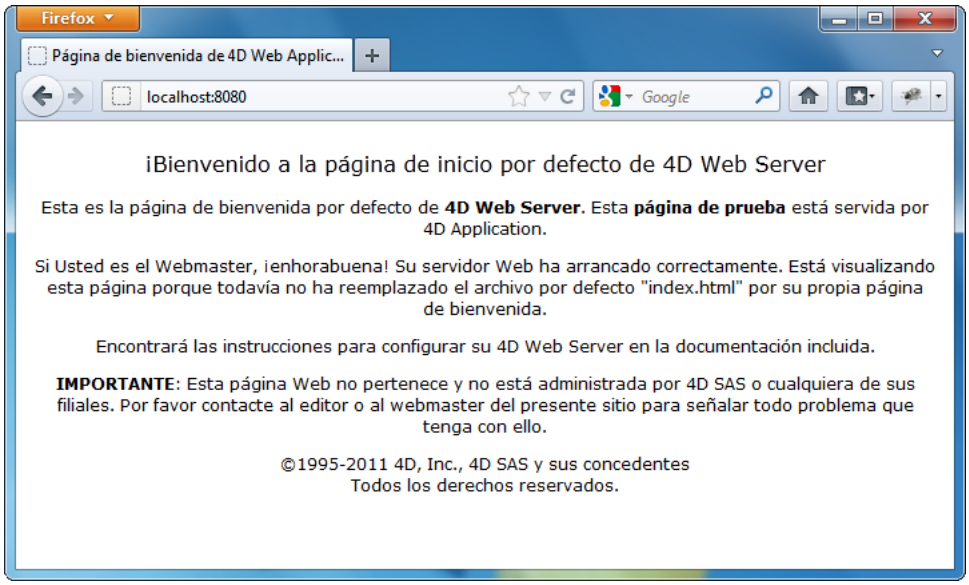
Raíz HTML por defecto

*Permite definir la ubicación por defecto de los archivos del sitio web e indica el nivel jerárquico en disco sobre el cual no se podrá acceder a los archivos. Esta opción se describe en la sección **Seguridad de las conexiones**.*

Página de inicio por defecto

Esta opción permite designar la página de inicio por defecto para todos los navegadores que se conectan a la base. Esta página puede ser estática o semidinámica.

Por defecto, cuando el servidor web se lanza por primera vez, 4D crea una página de inicio llamada “index.html” y la coloca en la carpeta raíz HTML. Si no modifica esta configuración, todo navegador que se conecte al servidor web obtendrá la siguiente página:



Para modificar la página web por defecto, puede reemplazarla simplemente por su propia página "index.html" en la carpeta raíz de la base o introducir la ruta de acceso relativa de la página que quiere definir el área "Página de inicio por defecto".

La ruta de accedo debe establecerse con relación a la carpeta raíz HTML por defecto.

Para asegurar la compatibilidad multiplataforma de sus bases, el servidor web 4D utiliza, para describir las rutas de acceso, convenciones de escritura particulares. Las reglas de sintaxis son las siguientes:

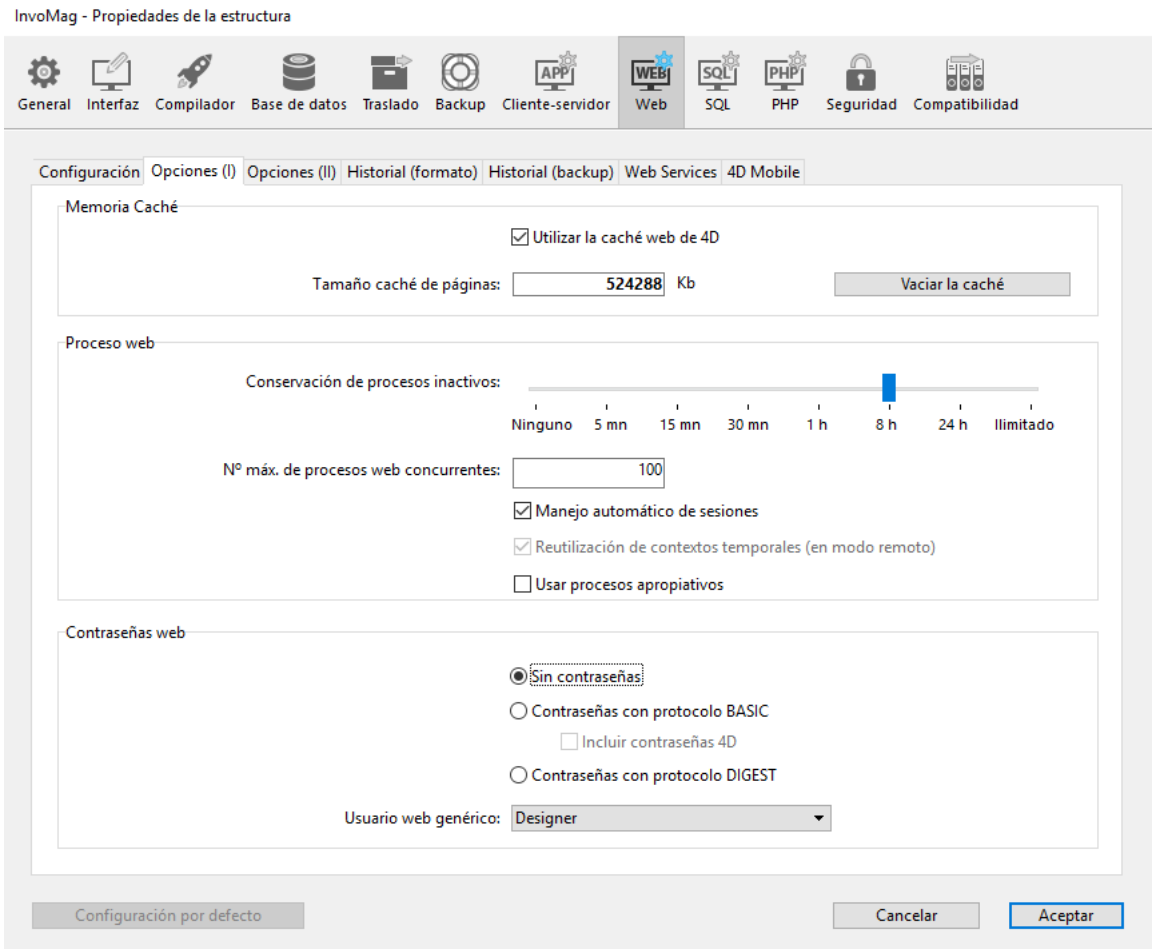
- *Las carpetas se separan por una barra oblicua ("/")*
- *La ruta de acceso no debe terminar con una barra oblicua ("/")*
- *Para "subir" un nivel en la jerarquía de la carpeta, introduzca ".." (Dos puntos) delante del nombre de la carpeta*
- *La ruta de acceso no debe comenzar con una barra oblicua ("/")*

Por ejemplo, si quiere que la página de inicio por defecto sea la página "MiCasa.htm", ubicada en la carpeta "Web" (ubicada en la carpeta raíz HTML de la base), introduzca "Web/MiCasa.htm".

***Nota:** también puede definir una página de inicio por defecto para cada proceso web utilizando la rutina **WEB SET HOME PAGE**.*

*Si no especifica una página de inicio por defecto, se llama al **Método de base On Web Connection**. Es su decisión procesar la petición por programación.*

Página Opciones (I)



Memoria Caché

El servidor web 4D dispone de una caché que permite cargar en memoria las páginas estáticas, las imágenes GIF, las imágenes JPEG (<512 kb) y las hojas de estilos (archivos .css), a medida en que son solicitadas.

La utilización de un caché permite aumentar de manera significativa el rendimiento del servidor web al enviar páginas estáticas.

La caché es común para todos los procesos web. Puede definir el tamaño de la caché en las Preferencias. Por defecto, la caché de las páginas estáticas no está activa. Para activarla, simplemente seleccione la opción ***Usar la caché Web de 4D****.*

Puede modificar el tamaño de la caché en el área ***Tamaño caché de páginas****. El valor a definir depende del número y tamaño de las páginas estáticas de su sitio web, como también de los recursos de que dispone el equipo local.*

Nota: *durante la utilización de su base web, puede controlar el desempeño de la caché utilizando la rutina* ***WEB GET STATISTICS****. Si por ejemplo, nota que la tasa de utilización de la caché es cercana al 100%, puede considerar aumentar el tamaño que se le ha adjudicado.*

Los URLs /4DSTATS y /4DHTMLSTATS también le permiten obtener información sobre el estado de la caché. Consulte la sección ***Información sobre el sitio web****.*

Una vez activada la caché, el servidor web 4D busca la página solicitada por el navegador primero en la caché. Si encuentra la página, la envía de inmediato. De lo contrario, 4D carga la página del disco y la ubica en la caché.

Cuando la caché está llena y se necesita espacio adicional, 4D "descarga" las páginas menos utilizadas, por orden de antigüedad.

Vaciar la caché

En cualquier momento, puede vaciar la caché de las páginas y de las imágenes que contiene (por ejemplo, si modifica una página estática y quiere cargarla en la caché).
Para hacer esto, tiene que hacer clic en el botón ***Vaciar caché****. La caché se vacía inmediatamente.*

Nota: *también puede utilizar la URL especial* ***/4DCACHECLEAR****.*

Timeout de Procesos web

Permite definir el tiempo de espera máximo antes de cerrar (timeout) de los procesos web inactivos en el servidor.

Número máximo de procesos web concurrentes

Esta opción indica el límite estrictamente superior de ***procesos web concurrentes*** *de todo tipo (procesos web estándar o pertenecientes al "grupo de procesos") que se pueden abrir simultáneamente en el servidor. Este parámetro permite la prevención de la saturación del servidor 4D como resultado de un número masivo de peticiones.*

Por defecto, este valor es 32 000. Puede definir el número entre 10 y 32 000.

Cuando se alcanza el número máximo de procesos web concurrentes (menos uno), 4D no crea más nuevos procesos y envía el siguiente mensaje "Servidor no disponible" (estado HTTP 503 – Servicio no disponible) a cada nueva petición.
Nota: *el número máximo de procesos web puede igualmente definirse utilizando el comando* ***WEB SET OPTION****.*

Acerca de la reserva de procesos web

La "reserva" de procesos web permite aumentar la reactividad del servidor web. Este grupo está dimensionado para un mínimo (0 por defecto) y un máximo (10 por defecto) de procesos a reciclar. Estos procesos pueden modificarse utilizando el comando ***SET DATABASE PARAMETER****. Una vez se cambie el número máximo de procesos web, si este número es inferior al límite superior de la "reserva", el limite se baja al número máximo de procesos web.*

¿Cómo determinar el valor a pasar?

En teoría, el número máximo de procesos web es el resultado de la división Memoria disponible/Tamaño de la pila de un proceso web. Otra solución es visualizar la información de los procesos web mostrada en el Explorador de ejecución: se indican el número actual de procesos web y el número máximo alcanzado desde el inicio del servidor web.

(*) El tamaño de la pila asignado por 4D para un proceso Web es alrededor de 512 KB para las versiones 64 bits y alrededor de 256 KB para las versiones de 32 bits (valores indicativos que pueden variar en función del contexto).

Gestión automática de las sesiones

Activa o desactiva el mecanismo interno para el control automático de sesiones de usuario por el servidor HTTP de 4D. Este mecanismo se describe en la sección ***Gestión de las sesiones web****.*

Por defecto, este mecanismo se activa en las bases de datos creadas a partir de 4D v13. Sin embargo, por razones de compatibilidad, está deshabilitado en bases de datos convertidas a partir de versiones anteriores de 4D. Usted debe habilitarlo explícitamente para beneficiarse de esta funcionalidad.

Cuando esta opción está seleccionada, la opción "Reutilización de los contextos temporales" se selecciona automáticamente (y bloqueada).

Reutilización de los contextos temporales (en modo remoto)

Permite optimizar el funcionamiento del servidor web de 4D en modo remoto reutilizando los procesos web creados para el procesamiento de peticiones web anteriores. De hecho, el servidor web de 4D necesita de un proceso web específico para la administración de cada petición web; en modo remoto, cuando es necesario, este proceso conecta al equipo 4D Server para acceder a los datos y al motor de la base de datos. Luego genera un contexto temporal utilizando sus propias variables, selecciones, etc. Una vez procesada la petición, el proceso se aborta.

Cuando la opción ***Reutilización de los contextos temporales*** *está seleccionada, en modo remoto 4D mantiene los procesos web específicos y los reutiliza para las peticiones posteriores. Al eliminar la etapa de creación del proceso, mejora el rendimiento del servidor web.*

En contraparte, debe asegurarse en este caso de inicializar sistemáticamente las variables utilizadas en métodos 4D para evitar obtener resultados incorrectos. Igualmente, es necesario borrar las selecciones o registros actuales definidos durante la petición anterior.
Notas:

- *Esta opción es seleccionada automáticamente (y bloqueada) cuando la opción* ***Gestión automática de sesiones*** *está seleccionada. De hecho, el mecanismo de gestión de sesiones está basado en el principio de reciclaje de los procesos web: cada sesión utiliza el mismo proceso que se mantiene durante la vida útil de la sesión.*

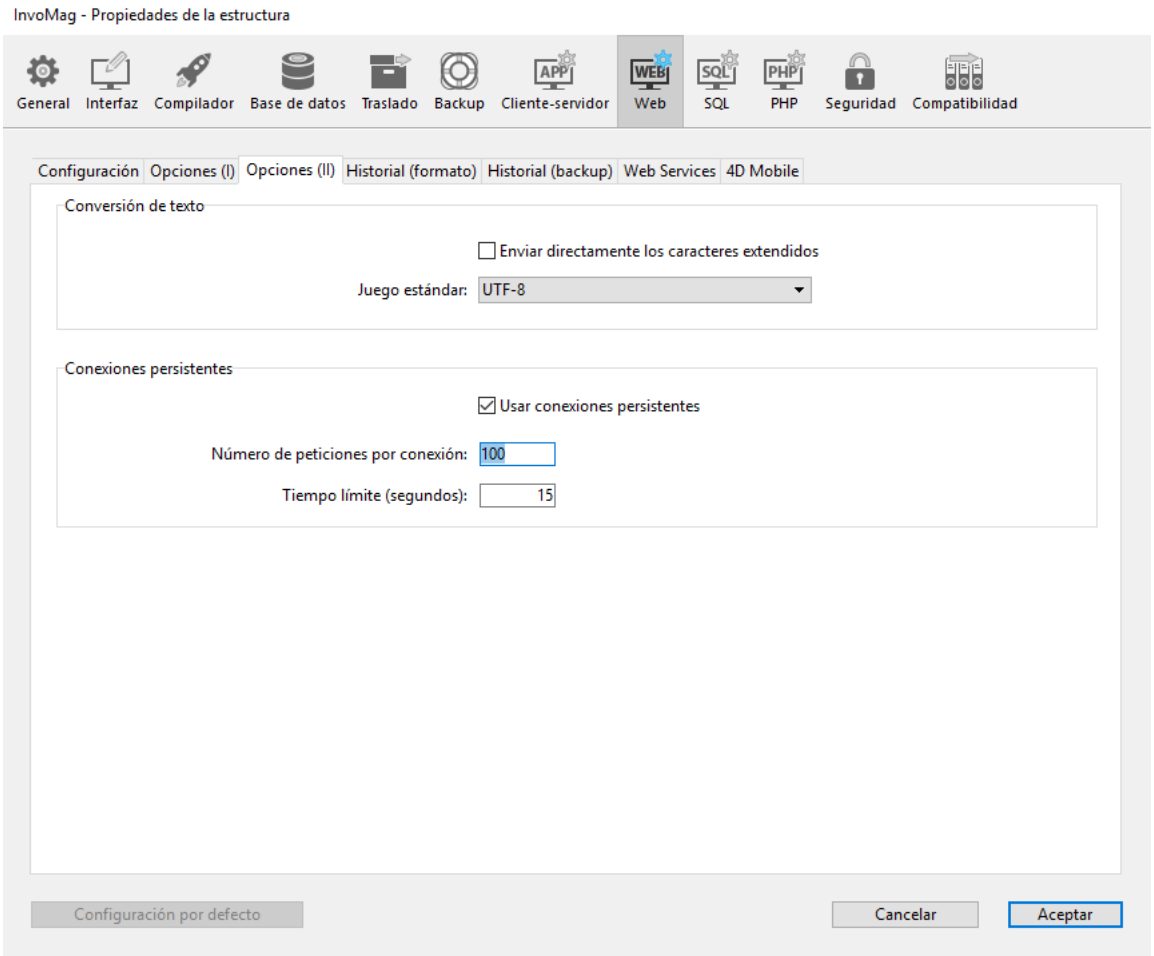
Sin embargo, tenga en cuenta que los procesos de la sesión no pueden ser "compartidos" entre diferentes sesiones: una vez terminada la sesión, el proceso se termina automáticamente (y no se reutiliza). Por tanto, es innecesario reiniciar las selecciones o variables, en este caso.

- *Esta opción sólo tiene efecto con un servidor web 4D en modo remoto. Con 4D en modo local, todos los procesos web (diferentes a los procesos de sesión) se eliminan después de ser utilizados.*

Área "Contraseñas"

La configuración del sistema de protección de acceso al sitio web utilizando contraseñas. Esta opción se describe en la sección ***Seguridad de las conexiones****.*

Página Opciones (II)



Enviar directamente los caracteres extendidos

Por defecto, el servidor web 4D convierte los caracteres extendidos presentes en las páginas web (dinámicas y estáticas) a los estándares HTML antes de enviarlas. Son interpretados por los navegadores.

Puede configurar el servidor web de manera que los caracteres extendidos sean enviados "tal cual", sin convertirlos a entidades HTML. Esta opción permite ganar una velocidad importante en los sistemas operativos extranjeros (especialmente en el sistema japonés).

Para hacer esto, seleccione la opción ***Enviar directamente los caracteres extendidos****.*

Juego estándar

La lista desplegable ***Juego estándar*** *permite definir el juego de caracteres estándar utilizado por el servidor web 4D. Por defecto, el juego de caracteres es UTF-8.*

Nota: *esta configuración también se utiliza para la generación de informes rápidos en formato HTML (ver* ***Ejecutar un informe rápido****).*

Conexiones persistentes

El servidor web de 4D puede utilizar conexiones persistentes. La opción de conexiones persistentes mantiene abierta una sola conexión TCP para el conjunto de cambios entre un navegador web y el servidor para economizar los recursos y optimizar las transferencias.

La opción ***Usar conexiones persistentes*** *activa o desactiva las conexiones TCP persistentes para el servidor web. Esta opción está seleccionada por defecto. En la mayoría de los casos, es recomendable conservar esta opción seleccionada ya que acelera los intercambios. Si el navegador web no soporta conexiones persistentes, el servidor web 4D automáticamente pasa a HTTP/1.0.*

La función conexiones persistentes del servidor web de 4D concierne a todas las conexiones TCP/IP (HTTP, HTTPS). Note sin embargo que las conexiones persistentes no siempre se utilizan para todos los procesos web 4D. En ciertos casos, pueden invocarse otras funciones de optimización del servidor web. Las conexiones persistentes son útiles principalmente durante el envío de páginas estáticas.

Dos opciones le permiten definir el mecanismo de conexiones persistentes:

● ***Número de peticiones por conexión****: permite definir el número máximo de peticiones y respuestas que pueden viajar en una misma conexión persistente. Limitar el número de peticiones por conexión permite evitar los riesgos de saturación del servidor vía el envío masivo de peticiones (técnica utilizada por los piratas).*

El valor por defecto (100) puede aumentarse o disminuirse dependiendo de los recursos de la máquina que aloja al servidor web 4D.

● ***Tiempo límite (segundos)****: este valor define el tiempo límite de espera (en segundos) durante el cual el servidor web mantiene abierta una conexión TCP sin recibir peticiones por parte del navegador web. Una vez terminado este periodo, el servidor cierra la conexión.*

Si el navegador web envía una petición después del cierre de la conexión, una nueva conexión TCP se crea automáticamente. Este funcionamiento es transparente para el usuario.

Utilizar procesos web apropiativos

El servidor Web integrado de 4D 64 bits para Windows y para OS X le permite sacar el máximo provecho de los ordenadores de varios núcleos mediante el uso de los procesos web apropiativos en sus aplicaciones compiladas. Puede configurar su código relacionado con la Web, incluyendo las etiquetas 4D y los métodos de base de datos Web, para funcionar simultáneamente en tantos núcleos como sea posible.

Para obtener más información sobre la funcionalidad de los procesos apropiativos en 4D, consulte la sección ***Procesos 4D apropiativos****.*

Disponibilidad del modo apropiativo para procesos Web

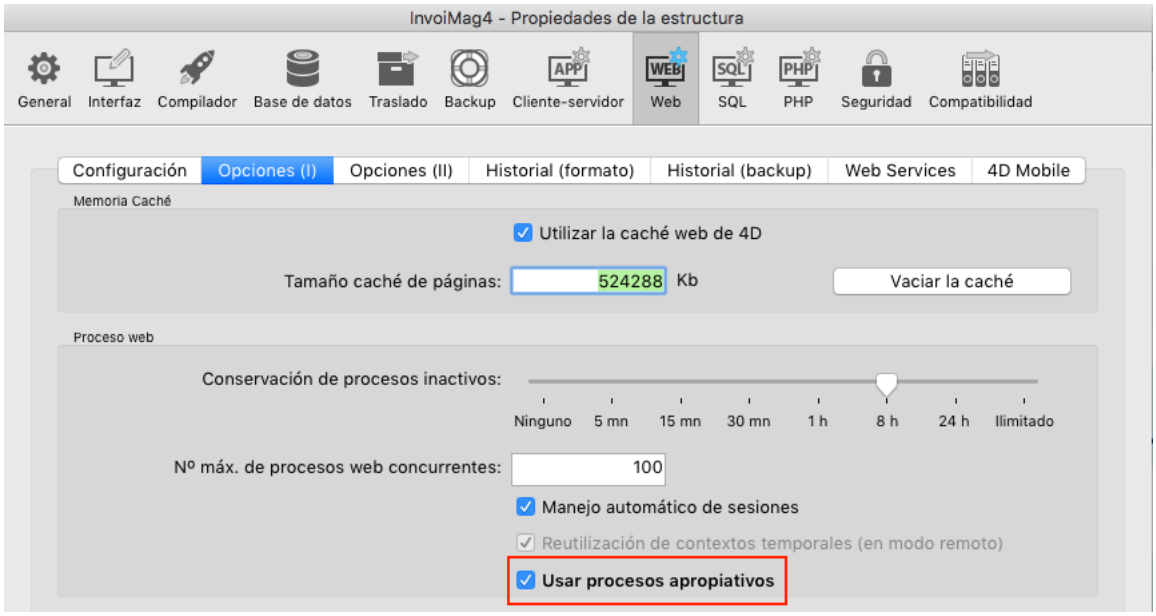
El uso del modo apropiativo para los procesos web sólo está disponible en los siguientes contextos:

- *uso de una versión de 64 bits de 4D*
- *uso de 4D Server o 4D modo local (4D en modo remoto no admite el modo apropiativo)*
- *uso de una base de datos compilada*
- *uso de la configuración de base de* ***procesos apropiativos*** *seleccionada (ver más adelante)*
- *todos los métodos de bases de datos relacionados con la Web y los métodos de proyecto se confirman hilo seguro por el Compilador 4D.*

Si algún requisito no se encuentra, el servidor web utilizará procesos cooperativos.

Activar el modo apropiativo para el servidor web

Para activar el modo preferente para el código del servidor Web de su aplicación, debe seleccionar la opción ***Utilizar los procesos apropiativos*** *en la página e"Web/Opciones (I)" del diálogo Propiedades de la base:*



Cuando se selecciona esta opción, el compilador 4D evaluará automáticamente la propiedad hilo seguro de cada código relacionado con la Web (ver abajo) y devuelve los errores en caso de incompatibilidad.

Escritura de código del servidor web hilo seguro

Todo el código 4D ejecutado por el servidor Web debe ser hilo seguro si desea que sus procesos web se ejecuten en modo apropiativo. Cuando la opción ***Utilizar procesos apropiativos*** *está seleccionada en las propiedades de la base, las siguientes partes de la aplicación serán evaluadas automáticamente por el compilador 4D:*

- *Todos los métodos base relacionados con la Web:*
  - ***Método base On Web Authentication***
  - ***Método base On Web Connection***
  - ***Método base On Web Close Process***
  - ***Método base On 4D Mobile Authentication***
- *Método proyecto* ***compiler_web*** *(cualquiera que sea su propiedad "Modo de ejecución");*
- *básicamente, cualquier código procesado por el comando* ***PROCESS 4D TAGS*** *en el contexto Web, por ejemplo a través de páginas .shtml.*
- *cualquier método de proyecto con el atributo "Disponible a través de etiquetas HTML 4D y URLS (4DACTION...)"*
- *disparadores para las tablas con el atributo "Exponer con Servicio 4D Mobile"*
- *métodos de proyecto disponibles a través de 4D Mobile (Propiedad "4D Mobile" seleccionada)*

Para cada uno de estos métodos y partes de código, el compilador comprobará si se respetan las normas hilo seguro, y devolverá errores en caso de problemas. Para más información sobre reglas hilo seguro, consulte el párrafo ***Escribir un método hilo seguro****.*

Seguridad de hilo de comandos 4D y URLs Web

A partir de 4D v16, la mayoría de los comandos 4D relacionados con la Web, métodos base y URLs son hilo seguro y se puede utilizar en el modo preventivo:

Comandos 4D

Todos los comandos relacionados con la Web 4D son hilos seguro, es decir:

- *todos los comandos del tema "****Servidor Web****",*
- *todos los comandos del tema "****Cliente HTTP****".*

Métodos base

Los siguientes métodos base son hilo seguro y se pueden utilizar en modo apropiativo:

- ***Método base On Web Authentication***
- ***Método base On Web Connection***
- ***Método base On Web Close Process***
- ***Método base On 4D Mobile Authentication***

Por supuesto, el código ejecutado por estos métodos también debe ser hilo seguro.

URLs servidor web

Los siguientes URLs del servidor web 4D son hilo seguro y pueden utilizarse en modo apropiativo:

- *4daction/ (el método de proyecto llamado debe también ser hilo seguro)*
- *4dcgi/ (los métodos base llamados debe también ser hilo seguro)*
- *4dscript/ (obsoleto como URL, utilizado como una etiqueta)*
- *4dwebtest/*
- *4dblank/*
- *4dstats/*
- *4dhtmlstats/*
- *4dcacheclear/*
- *rest/*
- *4dimgfield/ (generado por* ***PROCESS 4D TAGS*** *para petición web en campos imagen)*
- *4dimg/ (generado por* ***PROCESS 4D TAGS*** *para petición web en variables imagen)*

Los siguientes URLs de 4D Web Server no son hilo seguro y no son soportados en modo apropiativo:

- *4dsync*
- *4dsqlauth (obsoleto, utilizado para Flex 1.1)*

Icono de proceso web apropiativo

Tanto el Explorador de ejecución y la ventana de administración del servidor 4D muestran un icono específico para procesos web apropiativos:

Tipo de proceso	***Icono***
Método Web apropiativo	

Información sobre el sitio web

4D le permite obtener información sobre el funcionamiento de su sitio web 4D.

- *Puede controlar el sitio utilizando URLs particulares (/4DSTATS, /4DHTMLSTATS, /4DCACHECLEAR y /4DWEBTEST).*
- *Puede generar un historial de peticiones.*
- *Puede obtener información sobre el servidor web en la página Evaluación el Explorador de ejecución de 4D.*

***Nota:** por razones de seguridad, a partir de 4D v15 R2, el método HTTP TRACE está desactivado por defecto en el servidor web de 4D. Esto significa que cuando se recibe una petición HTTP TRACE, el servidor web de 4D ahora devuelve un error 405 ("método no autorizado"). Si desea habilitar explícitamente el método HTTP TRACE, debe utilizar la opción Web HTTP TRACE con el comando **WEB SET OPTION**.*

URLs de gestión del servidor web

El servidor web 4D acepta cuatro URLs particulares: /4DSTATS, /4DHTMLSTATS, /4DCACHECLEAR y /4DWEBTEST.
• /4DSTATS, /4DHTMLSTATS y /4DCACHECLEAR son accesibles únicamente al Diseñador y Administrador de la base. Si la base no tiene sistema de contraseñas, estos URLs están disponibles para todos los usuarios.
• /4DWEBTEST siempre está disponible.

/4DSTATS

El URL /4DSTATS devuelve diferente información en una tabla HTML (visible en un navegador):

- ***Cache Current Size**: tamaño actual de la caché del servidor web (en bytes)*
- ***Cache Max Size**: tamaño máximo de la caché (en bytes)*
- ***Cached Object Max Size**: tamaño máximo de cada objeto en la caché (en bytes)*
- ***Cache Use**: porcentaje de la caché utilizada*
- ***Cached Objects**: número de objetos (páginas, archivos imagen, etc.) presentes en la caché.*

*(*) Para mayor información sobre la caché de las páginas estáticas y de las imágenes, consulte la sección **Parámetros del servidor webParámetros del servidor web**.*

Esta información puede permitirle controlar el funcionamiento de su servidor y eventualmente adaptar los parámetros correspondientes.

***Nota:** el comando **WEB GET STATISTICS** también le permite obtener información sobre cómo la caché está siendo utilizada por las páginas estáticas.*

/4DHTMLSTATS

*El URL /4DHTMLSTATS devuelve, igualmente en forma de tabla HTML, la misma información que el URL /4DSTATS. La diferencia es que en el último campo **Cached Objects**, sólo cuenta las páginas HTML, sin los archivos imagen. Además, este URL devuelve el campo **Filtered Objects**.*

- ***Filtered Objects**: número de objetos de la caché no contabilizados por el URL, en particular las imágenes.*

/4DCACHECLEAR

El URL /4DCACHECLEAR provoca el borrado inmediato de la caché de las páginas estáticas y de las imágenes. Le permite por lo tanto "forzar" la actualización de las páginas que han sido modificadas.

/4DWEBTEST

El URL /4DWEBTEST permite controlar el estado del servidor web. Cuando se llama esta URL, 4D devuelve un archivo de texto que contiene únicamente los siguientes campos HTTP:

- ***Date:** fecha actual en el formato RFC 822*
  Por ejemplo: "Mon, 16 Jan 2012 13:12:50 GMT"
- ***Server:** 4D WebStar_D/número de versión interna*
  Por ejemplo: "4D WebStar_D/7.0"
- ***User-Agent:** nombre y versión @ dirección IP del cliente*
  Por ejemplo: "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:9.0.1) Gecko/20100101 Firefox/9.0.1 @ 127.0.0.1

Historial de las peticiones

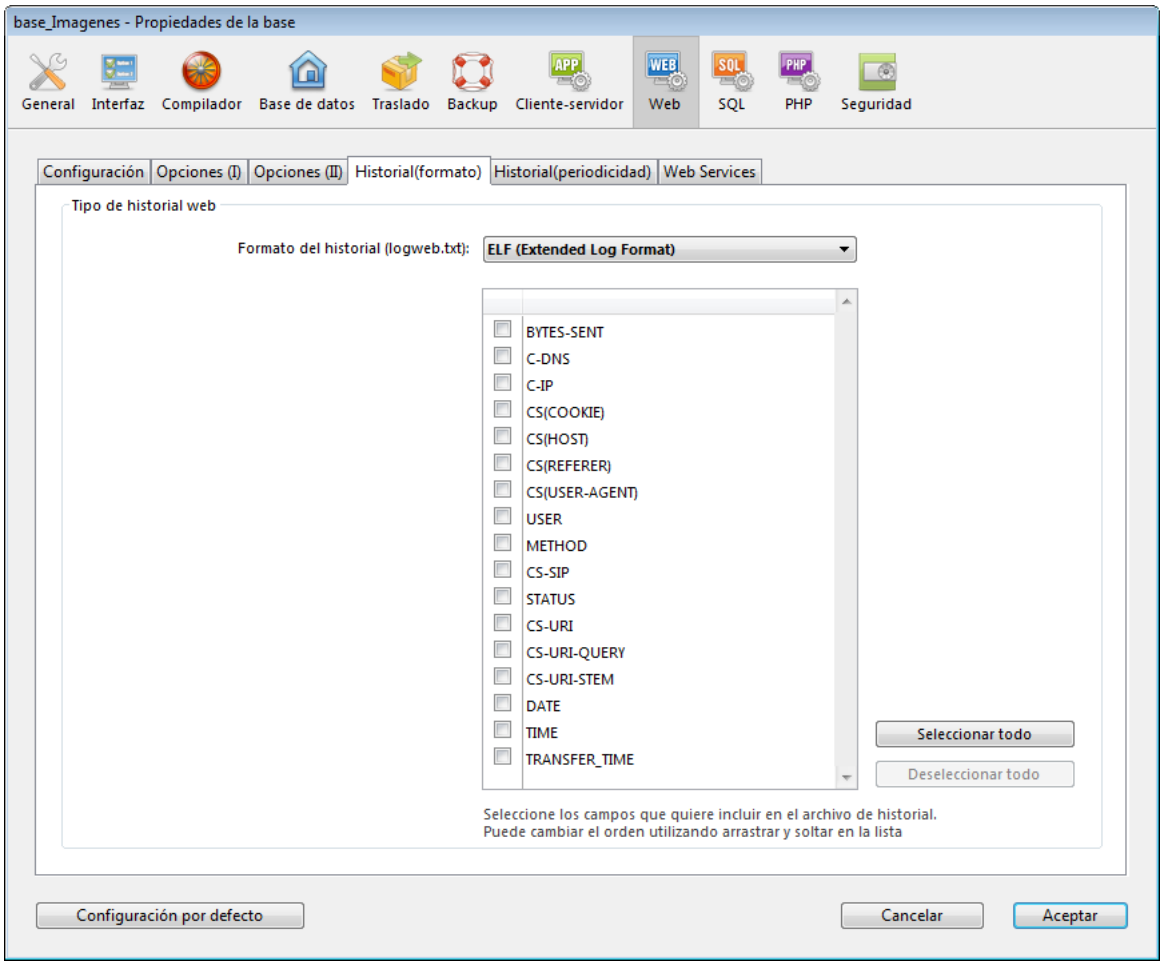
4D le permite obtener un historial de peticiones.

Este archivo se llama "logweb.txt" y se ubica automáticamente:

- *con 4D en modo local y 4D Server, en la carpeta **Logs** ubicada junto al archivo de estructura de la base.*
- *con 4D en modo remoto, en la subcarpeta **Logs** de la carpeta de base del cliente 4D (carpeta caché).*

Activación y formato

La activación y la configuración del contenido del archivo de historial se efectúa en las Propiedades de la base, página Web/Historial (formato):



Nota: *la activación y desactivación del archivo de historial de las peticiones puede igualmente realizarse por programación utilizando el comando* ***SET DATABASE PARAMETER*** *(4D v12) o* ***WEB SET OPTION*** *(4D v13 y posteriores).*

El menú de formato del log ofrece las siguientes opciones:

- ***Sin historial****: cuando esta opción está seleccionada, 4D no generará un archivo de historial de las peticiones.*
- ***CLF (Common Log Format)****: cuando se selecciona esta opción, el historial de peticiones se genera en formato CLF. Con el formato CLF, cada línea de archivo representa una petición, tal como:*
  host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "request" state length

  Cada campo está separado por un espacio y cada línea termina en la secuencia CR/LF (carácter 13, carácter 10).
- *host: dirección IP del cliente (ej. 192.100.100.10)*
- *rfc931: información no generada por 4D, es siempre - (un signo menos)*
- *usuario: nombre del usuario tal como se autentica, o de lo contrario - (signo menos). Si el nombre de usuario contiene espacios, serán reemplazados por _ (un guión bajo)*
- *DD: día, MMM: abreviación de 3 letras del nombre del mes (Ene, Feb,...), YYYY: año, HH: hora, MM: minutos, SS: segundos*
- *La fecha y hora son locales al servidor.*
- *request: petición enviada por el cliente (ej. GET /index.htm HTTP/1.0)*
- *state: respuesta dada por el servidor.*
- *length: tamaño de los datos devueltos (excepto el encabezado HTTP) ó 0.*

Nota: *por razones de rendimiento, las operaciones se guardan en un buffer de memoria en paquetes de 1Kb antes de ser escritas en el disco. Las operaciones también están escritas en el disco si ninguna petición ha sido enviado cada 5 segundos.*
Los posibles valores del estado son los siguientes:

200: OK
204: Sin contenido
302: Redirección
304: No modificado
400: Autenticación incorrecta
401: Autenticación necesaria
404: No encontrada
500: Error interno

El formato CLF no puede personalizarse.

- ***DLF (Combined Log Format)****: cuando se selecciona esta opción, el historial de peticiones se genera en el formato DLF. El formato DLF es similar al formato CLF y utiliza exactamente la misma estructura. Simplemente contiene dos campos HTTP adicionales al final de cada petición: Referer y User-agent.*

- Referer: contiene el URL de la página que apunta al documento solicitado.
- User-agent: contiene el nombre y la versión del navegador o del paquete cliente en el origen de la petición.

El formato DLF no puedes personalizarse.

- ***ELF (Extended Log Format)**: cuando esta opción está seleccionada, el historial de peticiones se genera en formato ELF. El formato ELF está ampliamente expandido en el mundo de los navegadores HTTP. Puede utilizarse para crear historiales sofisticados, que respondan a necesidades específicas. Por esta razón, el formato ELF es personalizable: es posible elegir los campos a grabar así como el orden de inserción en el archivo.*

- ***WLF (WebStar Log Format)**: cuando se selecciona esta opción, el historial de peticiones se genera en formato WLF. El formato WLF fue desarrollado específicamente por el servidor 4D WebSTAR. Es similar al formato ELF, con sólo unos pocos campos adicionales. Al igual que el formato ELF, es personalizable.*

Configurar los campos

Cuando elige el formato ELF (Extended Log Format) o WLF (WebStar Log Format), el área "Formato personalizado del historial Web" muestra los campos disponibles para el formato. Debe seleccionar cada campo a incluir en el historial. Para hacerlo, utilice los botones de flecha o simplemente arrastre y suelte los campos que quiere al área "Campos seleccionados".

***Nota**: no es posible seleccionar el mismo campo dos veces.*

La siguiente tabla lista los campos disponibles para cada formato (en orden alfabético) y describe sus contenidos:

Campo	***ELF***	***WLF***	***Valor***
BYTES_RECEIVED		*X*	*Número de bytes recibidos por el servidor*
BYTES_SENT	*X*	*X*	*Número de bytes enviados por el servidor al cliente*
C_DNS	*X*	*X*	*Dirección IP del DNS (ELF: campo idéntico al campo C_IP)*
C_IP	*X*	*X*	*Dirección IP del cliente (por ejemplo 192.100.100.10)*
CONNECTION_ID		*X*	*Número único de la conexión*
CS(COOKIE)	*X*	*X*	*Información sobre las cookies contenidas en al petición HTTP*
CS(HOST)	*X*	*X*	*Campo Host de la petición HTTP*
CS(REFERER)	*X*	*X*	*URL de la página que apunta al documento solicitado*
CS(USER_AGENT)	*X*	*X*	*Información sobre el software y el sistema operativo del cliente*
CS_SIP	*X*	*X*	*Dirección IP del servidor*
CS_URI	*X*	*X*	*URI en la cual la petición se efectúa*
CS_URI_QUERY	*X*	*X*	*Parámetros de búsqueda de peticiones*
CS_URI_STEM	*X*	*X*	*Parte de la petición sin los parámetros de búsqueda*
DATE	*X*	*X*	*DD: día, MMM: abreviación de 3 letras para el mes (Ene, Feb, etc.), YYYY: año*
METHOD	*X*	*X*	*Método HTTP utilizado para la petición enviada al servidor*
PATH_ARGS		*X*	*Parámetros de la CGI: cadena ubicada después del carácter "$"*
STATUS	*X*	*X*	*Respuesta ofrecida por el servidor*
TIME	*X*	*X*	*HH: hora, MM: minutos, SS: segundos*
TRANSFER_TIME	*X*	*X*	*Tiempo solicitado por el servidor para generar la respuesta*
USER	*X*	*X*	*Nombre del usuario si es autenticado; de lo contrario - (signo menos).* *Si el nombre de usuario contiene espacios, se reemplazan por _ (guiones bajos)*
URL		*X*	*URL solicitada por el cliente*

***Nota:** fechas y horas dadas en GMT.*

Periodicidad de los backups

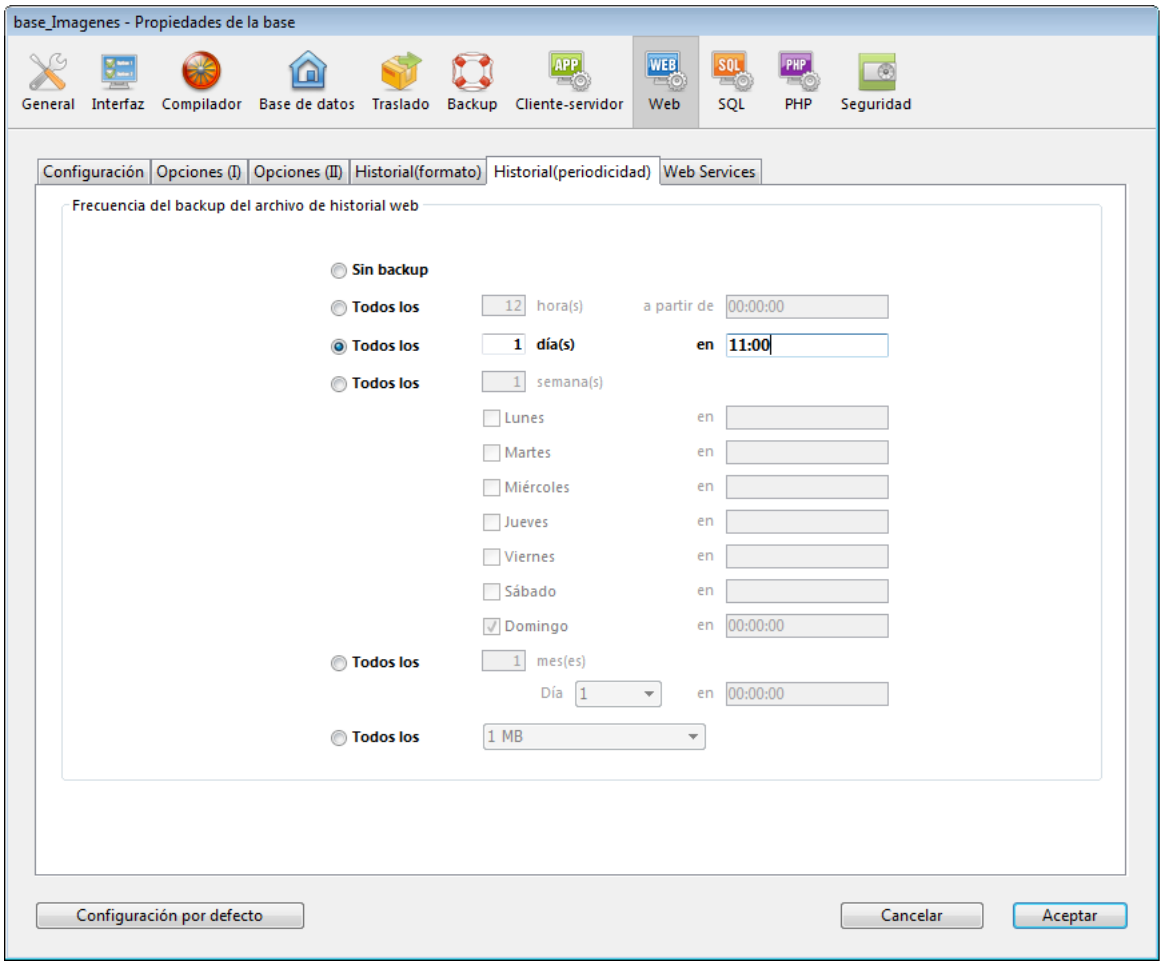
Como un archivo de historial de peticiones puede volverse de un tamaño considerable, es posible configurar un mecanismo de archivo automático. La activación de un backup puede estar basada en un cierto periodo de tiempo (expresado en horas, días, semanas o meses), o basado en el tamaño del archivo; cuando se alcanza el tiempo (o tamaño de archivo) definido, 4D se cierra y el archiva automáticamente el archivo de historial actual y crea uno nuevo.

Cuando se activa el backup del archivo de historial web, el archivo historial se guarda en una carpeta llamada "Logweb Archives," la cual se crea en el mismo nivel que el archivo logweb.txt (es decir, al lado del archivo de estructura de la base).

El archivo guardado se renombra basado en el siguiente ejemplo: "DYYYY_MM_DD_Thh_mm_ss.txt." Por ejemplo, para un archivo guardado el 4 de septiembre del 2006 a las 3:50 p.m. y 7 segundos: "D2006_09_04_T15_50_07.txt."

Parámetros de los backups

Los parámetros de backup automático del historial de peticiones se definen en la página Web/Historial (periodicidad) de las Propiedades de la base:



Primero debe elegir la frecuencia (días, semanas, etc.) o el criterio de tamaño limite del archivo haciendo clic en el botón correspondiente. Luego debe especificar el momento preciso para realizar el backup si es necesario.

- ***Sin backup**: la función de backup programado está desactivada.*
- ***Todos los X hora(s)**: esta opción se utiliza para programar backups a determinadas horas. Puede introducir un valor entre 1 y 24 .*
  *- **a partir de**: utilizado para definir la hora a la cual comenzará el primer backup.*
- ***Todos los X día(s) a las X**: esta opción se utiliza para programar backups diarios. Introduzca 1 si quiere realizar un backup diariamente. Cuando esta opción está seleccionada, debe indicar la hora a la cual debe comenzar el backup.*
- ***Todos los X semana(s), día a las X**: esta opción permite programar backups basados en semanas. Introduzca 1 si quiere realizar un backup semanalmente. Cuando esta opción está seleccionada, debe indicar el (los) día(s) de la semana y la hora a la cual el backup debe comenzar. Puede seleccionar varios días de la semana si lo desea. Por ejemplo, puede utilizar esta opción para definir dos backups a la semana: uno el miércoles y el otro los viernes.*
- ***Todos los X mes(es), X día a las X**: esta opción se utiliza para programar backups basados en meses. Introduzca 1 si quiere realizar un backup mensual. Cuando esta opción está seleccionada, debe indicar el día del mes y la hora a la que el backup debe comenzar.*
- ***Todos los X MB**: esta opción se utiliza para programar backups basados en el tamaño del archivo de historial de peticiones actual. Un backup se dispara automáticamente cuando el archivo alcanza el tamaño definido. Puede definir un tamaño límite de 1, 10, 100 o 1000 MB.*

***Nota:** en el caso de los backups programados, si el servidor web no fue lanzado en el momento del backup programado, en el siguiente inicio 4D considera el backup como fallido y aplica los parámetros apropiados, definidos en las Propiedades.*

Información del Explorador de ejecución de 4D

*La página **Evaluación** (título "Web") en el Explorador de ejecución muestra la información relativa al servidor web, particularmente:*

- ***Ocupación de la caché web**: indica el número de páginas presentes en la caché Web así como el porcentaje de utilización. Esta información sólo está disponible si el servidor web está activo y si el tamaño de la caché es mayor de 0.*
- ***Tiempo de actividad del servidor web**: indica la duración del funcionamiento (en formato horas:minutos:segundos) del servidor web. Esta información sólo está disponible si el servidor web está activo.*
- ***Número de peticiones HTTP**: indica el número total de peticiones HTTP recibidas desde el lanzamiento del servidor web, como también el número instantáneo de las peticiones por segundo (medida tomada entre dos actualizaciones del Explorador de ejecución). Esta información sólo está disponible si el servidor web está activo.*

***Nota:** para mayor información sobre el Explorador de ejecución, consulte el Manual de Diseño.*

Definir las páginas de errores HTTP personalizadas

El servidor web 4D le permite personalizar las páginas de error HTTP enviadas a los clientes, según el código de estado de la respuesta del servidor. Las páginas de error se refieren a:

- *códigos de estado que comienzan con 4 (errores del cliente), por ejemplo 404*
- *códigos de estado que comienzan con 5 (errores del servidor), por ejemplo 501.*

Para una descripción completa de los códigos de estado de error HTTP, puede consultar la Lista de códigos de estado HTTP (Wikipedia).

¿Cómo funciona?

Para reemplazar las páginas de error predeterminadas del Servidor Web 4D con sus propias páginas, solo necesita:

- *poner páginas HTML personalizadas en el primer nivel de la carpeta web de la aplicación,*
- *nombre las páginas personalizadas "{statusCode}.html" (por ejemplo, "404.html").*

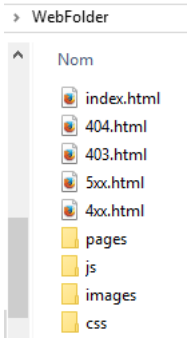
Puede definir una página de error por código de estado y/o una página de error genérica para un rango de errores, llamada "{número}xx.html". Por ejemplo, puede crear "4xx.html" para errores de clientes genéricos. El servidor web 4D primero buscará una página {codigoEstado}.html, y si no existe, una página genérica.

Por ejemplo, cuando una respuesta HTTP devuelve un código de estado 404:

1. *4D Web Server intenta enviar una página "404.html" ubicada en la carpeta web de la aplicación.*
2. *Si no se encuentra, 4D Web Server intenta enviar una página "4xx.html" ubicada en la carpeta web de la aplicación.*
3. *Si no se encuentra, 4D Web Server utiliza su página de error predeterminada.*

Ejemplo

Si define las siguientes páginas personalizadas en su carpeta web:



- *las páginas "403.html" o "404.html" se servirán cuando las respuestas HTTP 403 o 404 se devuelvan respectivamente,*
- *la página "4xx.html" se servirá para cualquier otro estado de error 4xx (400, 401, etc.),*
- *la página "5xx.html" se servirá para cualquier estado de error 5xx.*

Utilizar el protocolo TLS (HTTPS)

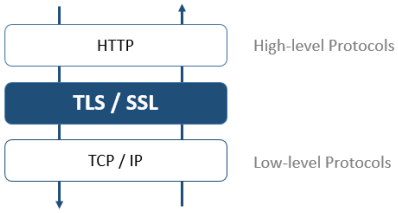
El servidor web 4D puede comunicarse en modo seguro vía el protocolo TLS (Secured Socket Layer), el sucedor de SSL (Secured Socket Layer). Por defecto, la versión mínima soportada en 4D es TLS 1.2.

Definición del protocolo TLS

El protocolo TLS (sucesor de SSL) ha sido diseñado para dar seguridad al intercambio de datos entre dos aplicaciones, principalmente entre un servidor web y un navegador. Este protocolo es ampliamente utilizado y es compatible con la mayoría de los navegadores web.

A nivel de la arquitectura de red, el protocolo de seguridad se inserta entre la capa TCP/IP (nivel bajo) y el protocolo de alto nivel HTTP, para el cual está destinado principalmente.

Configuración de red utilizando TLS:



***Nota:** el protocolo TLS también puede utilizarse para asegurar las conexiones cliente/servidor "clásicas" de 4D Server, así como también las conexiones del servidor SQL. Para mayor información, consulte la sección **Encriptar las conexiones cliente-servidor** en el manual de 4D Server y la sección **Configuración del servidor SQL de 4D** en el manual de referencia SQL.*

El protocolo TLS está diseñado para autenticar la identidad del emisor y del receptor, así como la confidencialidad e integridad de la información intercambiada:

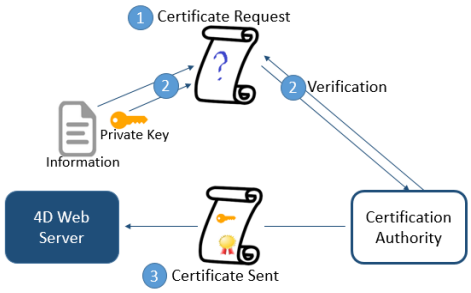
- ***Autenticación**: la identidad del emisor y receptor son confirmadas.*
- ***Confidencialidad**: los datos enviados se encriptan de manera que una tercera persona pueda entender el mensaje.*
- ***Integridad**: los datos recibidos no han sido alterados, por accidente o fraudulentamente.*

La llave privada se utiliza para encriptar los datos, es conservada por el emisor (el sitio web). La llave pública se utiliza para desencriptar la información y se envía a los receptores (navegadores web) por medio de un certificado. Cuando utiliza TLS en Internet, el certificado se entrega por medio de una autoridad de certificación, como Verisign®. El sitio web paga a la autoridad certificadores por entregar un certificado que garantice la autenticación del servidor y que contenga la llave pública que permita el intercambio de datos en modo seguro.
***Nota**: para mayor información sobre el método de encriptación y el empleo de llaves públicas/privadas, consulte la descripción del comando **ENCRYPT BLOB**.*

¿Cómo obtener un certificado?

Un servidor funcionando en modo seguro necesita un certificado numérico de una autoridad de certificación. Este certificado contiene información tal como el ID del sitio, así como también la llave pública utilizada para comunicarse con el sitio. Este certificado se transmite a los clientes (navegadores web) que se conectan al sitio. Una vez identificado y aceptado el certificado, se establece la comunicación en modo seguro.

***Nota**: un navegador acepta únicamente los certificados de una autoridad de certificación referenciada en sus propiedades.*



La elección de la autoridad de certificación depende de varios factores. Si la autoridad de certificación es conocida, el certificado será aceptado por muchos navegadores, sin embargo el precio será más alto.

Para obtener un certificado digital:

*1. Genere una llave privada utilizando el comando **GENERATE ENCRYPTION KEYPAIR**.*

***Advertencia:** por razones de seguridad, la llave privada siempre debe mantenerse secreta. De hecho, debe permanecer siempre en el equipo del servidor. Para el servidor web, el archivo **Key.pem** debe estar en la carpeta de la estructura de la base.*

*2. Utilice el comando **GENERATE CERTIFICATE REQUEST** para hacer una solicitud de certificado.*

3. Envíe la solicitud a la autoridad de certificación que haya elegido.
Para llenar la solicitud de certificación, necesita contactar la autoridad de certificación. La autoridad de certificación verifica que la información transmitida sea correcta. La solicitud del certificado se genera en un BLOB utilizando el formato PKCS codificado en base64 (formato PEM). Este principio autoriza copiar y pegar las llaves como texto y enviarlas vía e-mail con total seguridad sin modificar el contenido de la llave. Por ejemplo, puede guardar el BLOB que contiene la solicitud del certificado en un

*documento de texto (utilizando el comando **BLOB TO DOCUMENT**), luego abrir y copiar y pegar su contenido en un email o formulario web a enviar a la autoridad de certificación.*

4. Una vez haya recibido su certificado, cree un archivo de texto llamado "cert.pem" y pegue los contenidos del certificado en él.

Puede recibir un certificado de varias formas (generalmente por e-mail o un formulario HTML). El servidor web 4D acepta la mayoría de los formatos de texto (OS X, PC, Linux.). Sin embargo, el certificado debe estar en formato PEM, es decir PKCS, codificado en base 64.

***Nota:** los caracteres de fin de línea CR no están soportados; debe utilizar CRLF o LF.*

5. Ubique el archivo "cert.pem" en la ubicación adecuada. Para el servidor web, es la carpeta que contiene la estructura de la base.
Ahora el servidor web puede funcionar en modo seguro. Un certificado es válido por lo general entre seis meses y un año.

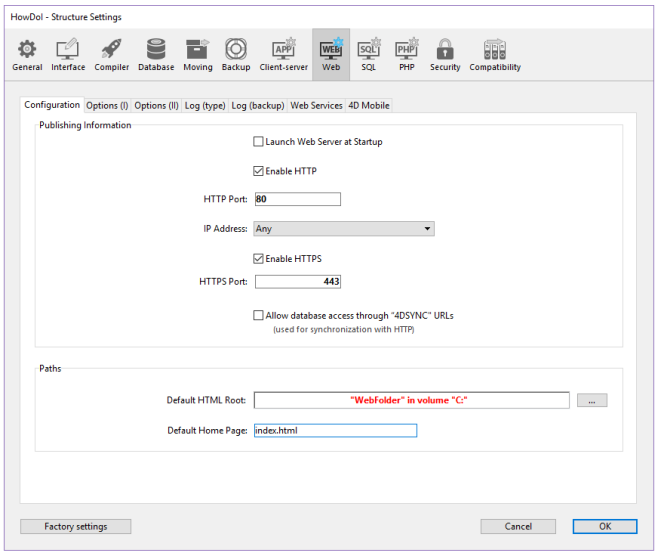
Instalación y activación del TLS en 4D

Para que pueda utilizar el protocolo TLS con el servidor web de 4D, varios elementos deben estar instalados en el servidor, en diferentes ubicaciones:

- *4DSLI.DLL (Windows) o 4DSLI.bundle (Mac OS): Interfaz de la capa segura dedicada a la gestión del TLS.*
  Este archivo se instala por defecto en:
  - Bajo Windows, junto al archivo ejecutable de la aplicación 4D o 4D Server.
  - Bajo Mac OS, en la subcarpeta Native Components del paquete 4D o 4D Server.
  ***Nota:** 4DSLI también es necesario para utilizar los comandos de encripción **ENCRYPT BLOB** y **DECRYPT BLOB**.*

- *key.pem: (documento que contiene la llave de encripción privada) y cert.pem (documento que contiene el certificado):*
  - con 4D en modo local o 4D Server, estos archivos deben estar en la carpeta de estructura de la base.
  *- con 4D en modo remoto, esto archivos deben estar en la carpeta de recursos locales de la base en el equipo remoto (para mayor información sobre la ubicación de esta carpeta, consulte el párrafo Carpeta base 4D Client en la descripción del comando **Get 4D folder**). Debe copiar los archivos manualmente en el equipo remoto.*

***Nota:** los archivos predeterminados key.pem y cert.pem se suministran con 4D. Para un mayor nivel de seguridad, le recomendamos que reemplace estos archivos con sus propios certificados.*
*La instalación de estos elementos hace posible la utilización del TLS en las conexiones al servidor web 4D. Sin embargo, para que las conexiones TLS sean aceptadas por el servidor web 4D, debe "activarlas" activando HTTPS. Este parámetro es accesible en la página **Web**, pestaña **Configuración** de las Propiedades de la base:*



Por defecto, las conexiones TLS están activas. Pude deseleccionar la opción HTTPS si no quiere utilizar las funcionalidades TLS con su servidor web, o si otro servidor web que permite conexiones seguras está operando en el mismo equipo.

*El puerto HTTPS dedicado para las conexiones TLS es por defecto el 443. Este número de puerto puede modificarse en el área **Número de puerto HTTPS**, por ejemplo, para reforzar la seguridad del servidor web (para mayor información sobre este punto, consulte la sección **Parámetros del servidor web**). El puerto HTTP definido en esta página de la propiedades se utiliza para las conexiones del servidor web en modo estándar.*

***Nota**: las otras propiedades de gestión del servidor web 4D (contraseña, tiempo límite, tamaño de la caché, etc.) se aplican, sin importar si el servidor está operando en modo TLS o no.*

Perfect Forward Secrecy (PFS)

PFS agrega una capa adicional de seguridad a sus comunicaciones. En lugar de utilizar llaves de intercambio preestablecidas, PFS crea llaves de sesión cooperativamente entre las partes que se comunican utilizando algoritmos Diffie-Hellman (DH). La manera conjunta en que se construyen las llaves crea un "secreto compartido" que impide que las partes externas puedan comprometerlas.

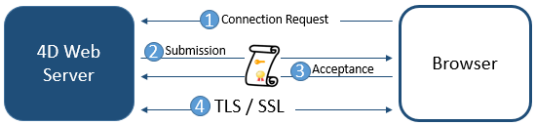
*Cuando TLS está activado en el servidor web 4D, PFS se habilita automáticamente. Si el archivo dhparams.pem (documento que contiene la llave privada DH del servidor) aún no existe, 4D lo generará automáticamente con un tamaño de llave de 2048. La generación inicial de este archivo podría demorar varios minutos. El archivo se coloca con los archivos key.pem y cert.pem, como se discutió en la sección anterior (**Instalación y activación del TLS en 4D**).*

Si usa una lista de cifrado personalizada y desea habilitar PFS, debe verificar que contenga entradas con algoritmos DH o ECDH (curva elíptica Diffie-Hellman).

***Nota:** para más información, consulte el selector SSL cipher list del comando **SET DATABASE PARAMETER**.*

Conexión de los navegadores con TLS

Para que una conexión Web se efectúe en modo seguro, simplemente el URL enviado por el navegador debe comenzar por "https" (en lugar de "http").
En este caso, aparece un diálogo de advertencia en el navegador. Si el usuario hace clic en ***OK****, el servidor Web envía el certificado al navegador.*



El algoritmo de encripción utilizado para la conexión es decidido por el navegador y el servidor Web. El servidor ofrece varios algoritmos de encriptación simétricos. Se utiliza el algoritmo común más poderoso.

Advertencia: *el nivel de encriptación permitido depende de las leyes actuales del país de uso.*

Control del modo de conexión

La utilización de TLS en el servidor web 4D necesita que se marque la casilla "Activar HTTPS" en el dialogo de configuración de la base. Esto asegura que las conexiones con el servidor web pueden ocurrir en un modo seguro. Sin embargo, tenga en cuenta que si la casilla de selección "Activar HTTP" también está marcada, el navegador puede cambiar entre HTTPS y HTTP (por ejemplo, en el área URL del navegador, el usuario puede reemplazar "https" por "http").

Notas*:*

- *Puede obtener el modo de conexión actual utilizando el comando* ***WEB Is secured connection****.*
- *La activación de HTTP y/o HTTPS también se puede establecer para la sesión utilizando el comando* ***WEB SET OPTION****.*

Puede prohibir o redirigir las solicitudes realizadas en un modo no seguro:

- *Para prohibir las redirecciones HTTP, desmarque la opción "Activar HTTP" (o utilice el selector correspondiente* ***WEB SET OPTION****). En este caso, el servidor web 4D ignorará las solicitudes HTTP del cliente y se mostrará un mensaje de error.*
- *Para redirigir automáticamente las solicitudes HTTP a HTTPS, puede activar HTTP Strict Transport Security (HSTS) utilizando el comando* ***WEB SET OPTION*** *con el selector Web HSTS enabled. Esto exigirá el uso de TLS y garantizará que todas las comunicaciones se realicen a través de HTTPS. Consulte la descripción del comando* ***WEB SET OPTION*** *para obtener información detallada sobre HSTS.*

Soporte de XML y WML

WML

El servidor web 4D soporta la tecnología WML (Wireless Markup Language). Esta funcionalidad autoriza la consulta y la entrada de información en las bases 4D con la ayuda de un teléfono celular o un asistente electrónico personal (PDA).

***Nota:** el lenguaje WML, asociado al protocolo WAP (Wireless Application Protocol), es desarrollado por varias empresas. La tecnología WAP ofrece un conjunto de herramientas de comunicación de redes de manera que los teléfonos celulares y las PDAs puedan visualizar texto publicado en las páginas Web. La tecnología WML es abierta y libre de derechos. Para mayor información sobre WML, por favor consulte el sito Web de Phone.com: http://www.phone.com/.*

Los datos pueden introducirse o leerse por medio de páginas WML utilizando etiquetas 4DTEXT o 4DSCRIPT.

Esta es la lista de documentos WML aceptados por el servidor Web 4D:

Extensión	Tipo MIME	Descripción
.wml	text/vnd.wap.wml	Páginas WML (siempre soportadas por 4D*)
.wmls	text/vnd.wap.wmlscript	Scripts WML (del lado del cliente)
.wmlc	application/vnd.wap.wmlc	Páginas WML binarias
.wmlsc	application/vnd.wap.wmlscript	Scripts WML binarios
.wbmp	picture/vnd.wap.wbmp	Imágenes bitmap para teléfonos celulares (no siempre soportado)

** Permite la inserción dinámica de datos vía las etiquetas 4DTEXT y 4DSCRIPT.*

XML

El servidor web 4D soporta los documentos .xml,.xls y .dtd los cuales se envían respectivamente con el tipo MIME "text/xml", "text/xsl" y "text/xml".

4D analiza su contenido y procesa sus etiquetas de tipo 4DTEXT o 4DSCRIPT (si las hay) para generar el XML dinámico.

WEB CLOSE SESSION

WEB CLOSE SESSION (idSesion)

Parámetro	***Tipo***		***Descripción***
idSesion	*Texto*	➡	*UUID de sesión*

Descripción

*El comando **WEB CLOSE SESSION** invalida la sesión existente designada por el parámetro idSesion. Si la sesión no existe, el comando no hace nada.*

Cuando este comando se llama desde un proceso web o desde cualquier otro proceso:

- *la fecha de vencimiento de la cookie está definida en 0,*
- *el **Método base On Web Close Process** se llama, permitiéndole guardar la información de la sesión,*
- *las selecciones se borran, los registros se desbloquean y las variables se reinician.*

Después de la ejecución de este comando, si un cliente web envía una petición con una cookie invalida, se abre una nueva sesión con una nueva cookie y se envía.

***Nota:** en el contexto de una sesión 4D Mobile, el comando **WEB CLOSE SESSION** cierra la sesión 4D Mobile cuyo ID se pasa en el parámetro IDsesion. Como una sesión 4D Mobile puede gestionar varios procesos, este comando solicita a todos los procesos Web relacionados con la sesión para terminar su ejecución.*

WEB GET BODY PART

WEB GET BODY PART (parte ; contenido ; nombre ; tipoMime ; nomArchivo)

Parámetro	Tipo		Descripción
parte	Entero largo	➡	Número de parte
contenido	BLOB, Texto	⬅	Contenido de la parte
nombre	Texto	⬅	Nombre de la variable "input"
tipoMime	Texto	⬅	Tipo mime del archivo
nomArchivo	Texto	⬅	Nombre del archivo enviado

Descripción

*El comando **WEB GET BODY PART** llamado en el contexto de un proceso web, permite analizar la parte "body" de una petición multi-part.*

*En el parámetro parte, pase el número de la parte a analizar. Puede obtener el número total de partes con el comando **WEB Get body part count**.*

El parámetro contenido recibe el contenido de la parte. Cuando las partes a recuperar son archivos, debe pasar un parámetro de tipo BLOB. En el caso de variables TEXTO enviadas en un formulario web, puede pasar un parámetro de tipo texto.

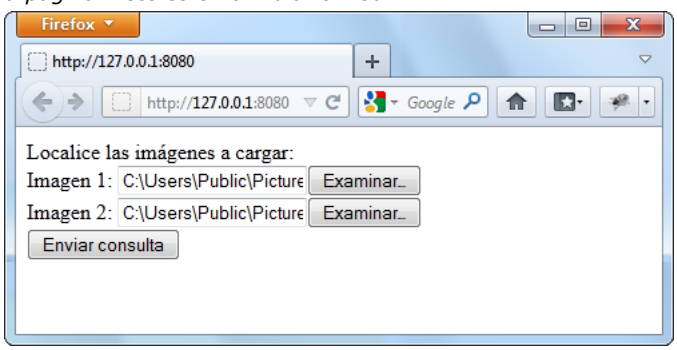
El parámetro nombre recupera el nombre de la variable del campo input HTTP.

*Los parámetros tipoMime y nomArchivo reciben el tipo Mime y el nombre del archivo original, si lo hay. nomArchivo solo se recibe cuando el archivo se envio como **<input type="file">**.*
tipoMime y nomArchivo son opcionales pero deben pasarse juntos.

***Nota**: en el contexto de una petición multi-part, el primer de array del comando **WEB GET VARIABLES** devuelve todas las partes del formulario, en el mismo orden que el comando **WEB GET BODY PART**. Puede utilizarlo para obtener directamente la posición de una parte de un formulario.*

Ejemplo

En este ejemplo, un formulario web permite descargar en el servidor HTTP varias imágenes desde un navegador y mostrarlas en la página. Este es el formulario web:



Este es el código para la parte <body> de la página:

```
<body> <form enctype="multipart/form-data" action="/4DACTION/GetFile/" method="post"> Localice las imágenes a cargar: <br> Imagen 1: <input name="file1" type="file"><br> Imagen 2: <input name="file2" type="file"><br> <input type="submit"> </form> <hr/> <!--4DSCRIPT/galleryInit--> <!--4Dloop aFileNames--> <img src="/photos/<!--4Dvar aFileNames{aFileNames}-->"/> <!--4Dendloop--> </body>
```

Dos métodos 4D son llamados por la página:

- ***galleryInit** al cargar (etiqueta 4DSCRIPT), muestra las imágenes presentes en la carpeta de destino.*
- ***GetFile** al enviar los datos (4DACTION URL del formulario multi-part), procesa el envío.*

*Este es el código del método **galleryInit**:*

```
C_TEXT($vDestinationFolder)
ARRAY TEXT(aFileNames;0)
C_LONGINT($i)
$vDestinationFolder:=Get 4D folder(HTML Root folder)+"photos"+Folder separator  //Carpeta "WebFolder/photos"
DOCUMENT LIST($vDestinationFolder;aFileNames)
```

*Este es el código del método **GetFile**:*

```
C_TEXT($vPartName;$vPartMimeType;$vPartFileName;$vDestinationFolder)
C_BLOB($vPartContentBlob)
C_LONGINT($i)
$vDestinationFolder:=Get 4D folder(HTML Root folder)+"photos"+Folder separator
For($i;1;WEB Get body part count) //para cada parte
    WEB GET BODY PART($i;$vPartContentBlob;$vPartName;$vPartMimeType;$vPartFileName)
    If($vPartFileName#"")
        BLOB TO DOCUMENT($vDestinationFolder+$vPartFileName;$vPartContentBlob)
    End if
End for
WEB SEND HTTP REDIRECT("/") // volver a la página
```

WEB Get body part count

WEB Get body part count -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Número de partes en el cuerpo

Descripción

*El comando **WEB Get body part count** devuelve el número de partes que compone el cuerpo recibido.*

Ejemplo

*Consulte el ejemplo del comando **WEB GET BODY PART**.*

WEB Get Current Session ID

WEB Get Current Session ID -> Resultado

Parámetro	***Tipo***	***Descripción***
Resultado	*Texto*	*UUID de la sesión*

Descripción

*El comando **WEB Get Current Session ID** devuelve el ID de la sesión abierta para la petición web. Esta identificación es generada automáticamente por 4D como un UUID.*

Si este comando se llama fuera del contexto de una sesión web, devuelve una cadena vacía "".

WEB GET HTTP BODY

WEB GET HTTP BODY (cuerpo)

Parámetro	Tipo		Descripción
cuerpo	BLOB, Texto	⇦	Cuerpo (Body) de la petición HTTP

Descripción

*El comando **WEB GET HTTP BODY** devuelve el cuerpo de la solicitud HTTP que esta siendo procesada. El cuerpo HTTP se devuelve tal cual, sin proceso ni análisis.*

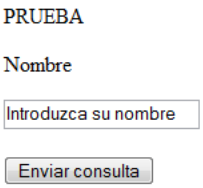
*Este comando puede llamarse utilizando un método de base web (**Método de base On Web Authentication**, **Método base On Web Connection**) o todo método web.*

Puede pasar en el parámetro cuerpo, una variable o un campo de tipo BLOB o Texto. El tipo Texto, por lo general será suficiente (el parámetro cuerpo puede recibir hasta 2GB de texto)

Este comando permite por ejemplo efectuar las búsquedas en el cuerpo de las solicitudes. También permite a los usuarios avanzados configurar un servidor WebDAV dentro de una base 4D.

Ejemplo

En este ejemplo, una solicitud simple se envía al servidor web de 4D y el contenido del campo HTTP cuerpo se visualiza en el depurador. Este es el formulario enviado al servidor web de 4D, así como también el código HTML correspondiente:




```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>P&aacute;gina de prueba</title>
</head>

<body>
PRUEBA
<form action="/4DACTION/Test4D2004" method="POST">
<p>Nombre</p>
<input type="text" value="Introduzca su nombre" name="vNombre" />
<br />
<br />
<input type="submit" value="Enviar consulta" />
</form>

</body>
</html>
```

Este es el método Test4D2004:

```
C_BLOB($peticion)
C_TEXT($textoPeticion)

WEB GET HTTP BODY($peticion)
$textoSolicitud:=BLOB to text($peticion;UTF8 text without length)
WEB SEND FILE("pagina.html")
```

***Nota:** este método se declara "Disponible vía las etiquetas HTML y los URLs 4D (4DACTION...)" en sus propiedades.*
Cuando el formulario se envia al servidor web, la variable $textoSolicitud recibe el texto del campo cuerpo de la petición HTTP.

WEB GET HTTP HEADER

WEB GET HTTP HEADER (encab|arrayCamp {; arrayValores})

Parámetro	Tipo		Descripción
encab|arrayCamp	Texto, Array texto	⬅	Encabezado HTTP de la petición o Campos del encabezado HTTP
arrayValores	Array texto	⬅	Contenido de los campos del encabezado HTTP

Descripción

*El comando **WEB GET HTTP HEADER** devuelve, en una cadena o dos arrays, el encabezado HTTP de la petición en proceso.*
*Este comando puede llamarse desde cualquier método (**Método de base On Web Authentication**, **Método base On Web Connection**, método llamado por "/4DACTION"...) ejecutado en un proceso web.*

- ***Primera sintaxis: WEB GET HTTP HEADER (encabezado)***

Cuando se utiliza esta sintaxis, el resultado devuelto en la variable encabezado es el siguiente:

```
"GET /page.html HTTP\1.0"+Char(13)+Char(10)+"User-Agent: browser"+Char(13)+Char(10)+"Cookie: C=HELLO"
```

Cada campo de encabezado está separado por una secuencia CR+LF (Retorno de carro+Retorno de línea) bajo Windows y Mac OS.

- ***Segunda sintaxis: WEB GET HTTP HEADER (arrayCamp; arrayValores)***

Cuando utiliza esta sintaxis, los resultados devueltos en los arrays arrayCamp y arrayValores son del siguiente tipo:

fieldArray{1} = "X-METHOD"	valueArray{1} = "GET" *
fieldArray{2} = "X-URL"	valueArray{2} = "/page.html" *
fieldArray{3} = "X-VERSION"	valueArray{3} = "HTTP/1.0" *
fieldArray{4} = "User-Agent"	valueArray{4} = "browser"
fieldArray{5} = "Cookie"	valueArray{5} = "C=HELLO"

** Estos tres primeros elementos no corresponden a los campos HTTP. Forman parte de la primera línea de la petición.*
Conforme al estándar HTTP, los nombres de los campos siempre se escriben en inglés.
Esta es una lista de algunos campos HTTP que pueden utilizarse en una petición:

- ***Accept**: contenido permitido por el navegador.*
- ***Accept-Language**: idioma(s) aceptado(s) por el navegador (para información). Permite seleccionar una página web utilizando el idioma definido en el navegador.*
- ***Cookie**: lista de cookies*
- ***From**: dirección de correo electrónico del usuario del navegador.*
- ***Host**: nombre o dirección del servidor (por ejemplo utilizando un URL, http://miservidorweb/mipagina.html, Host toma el valor «miservidorweb»). Permites administrar varios nombres que apuntan a la misma dirección IP (virtual hosting).*
- ***Referer**: origen de la petición (por ejemplo http://miservidorweb/mipagina1.html), es decir la página que el usuario muestra cuando se hace clic en el botón Anterior.*
- ***User-Agent**: nombre y versión del navegador o del proxy.*

Ejemplo

El siguiente método permite recuperar el contenido de todo campo de encabezado de petición HTTP:

```
` Método de proyecto GetHTTPField
` GetHTTPField (Text) -> Text
` GetHTTPField (Nombre encabezado HTTP) -> Contenido encabezado HTTP

C_TEXT($0;$1)
C_LONGINT($vlElem)
ARRAY TEXT($nombres;0)
ARRAY TEXT($valores;0)
$0:=""
WEB GET HTTP HEADER($nombres;$valores)
$vlElem:=Find in array($nombres;$1)
If($vlElem>0)
   $0:=$valores{$vlElem}
End if
```

- *Una vez escrito este método de proyecto, puede llamarse de esta forma:*

```
` Contenido del encabezado Cookie
$cookie:=GetHTTPField("Cookie")
```

- *Puede enviar diferentes páginas en función del idioma del navegador (por ejemplo en el **Método On Web Connection**):*

```
$idioma:=GetHTTPField("Accept-Language")
Case of
   :($idioma="@fr@") `Francés (ver lista ISO 639)
      WEB SEND FILE("index_fr.html")
   :($idioma="@sp@") `Español (ver lista ISO 639)
      WEB SEND FILE("index_es.html")
   Else
      WEB SEND FILE("index.html")
End case
```

Nota: *los navegadores web permiten definir varios idiomas por defecto. Están listados en el campo "Accept-Language", separados por un ";". Su propiedad está definida de acuerdo a su posición dentro de la cadena; por lo tanto es una buena idea probar la posición de los idiomas en la cadena.*

- *Este es un ejemplo de hosts virtuales (por ejemplo en el **Método base On Web Connection**). Los siguiente nombres "home_site.com", "home_site1.com" y "home_site2.com" apuntan a la misma dirección IP, por ejemplo 192.1.2.3.*

```
$host:=GetHTTPField("Host")
Case of
   :($host="www.site1.com")
      WEB SEND FILE("home_site1.com")
   :($host="www.site2.com")
      WEB SEND FILE("home_site2.com")
   Else
      SEND HTML FILE("home_site.com")
End case
```

WEB GET OPTION

WEB GET OPTION (selector ; valor)

Parámetro	*Tipo*		*Descripción*
selector	*Entero largo*	➡	*Código de la opción a modificar*
valor	*Entero largo, Texto*	⬅	*Valor de la opción*

Descripción

*El comando **WEB GET OPTION** permite leer el valor actual de una opción de funcionamiento del servidor web de 4D.*
*El parámetro selector indica la opción web a leer. Pase en este parámetro una constante del tema **Servidor web**:*

Constante	Tipo	Valor	Comentario
Web character set	*Entero largo*	*17*	***Alcance**: 4D local, 4D Server* ***Descripción**: conjunto de caracteres que el servidor Web 4D (con 4D en modo local y 4D Server) utiliza para comunicarse con los navegadores web que se conectan a base. El valor por defecto depende del lenguaje del sistema operativo. Este parámetro se define en las Propiedades de la base.* ***Valores**: los valores posibles dependen del modo de ejecución de la base relativos al conjunto de caracteres.* • *Modo Unicode: cuando la aplicación se ejecuta en modo Unicode, los valores a pasar para este parámetro son los identificadores de conjunto de caracteres (MIBEnum longint o Name string, identificadores definidos por IANA, consulte: http://www.iana.org/assignments/character-sets). Está es la lista de los identificadores correspondientes a los conjuntos de caracteres que admite el servidor Web de 4D:* *4=ISO-8859-1* *12=ISO-8859-9* *13=ISO-8859-10* *17=Shift-JIS* *2024=Windows-31J* *2026=Big5* *38=euc-kr* *106=UTF-8* *2250=Windows-1250* *2251=Windows-1251* *2253=Windows-1253* *2255=Windows-1255* *2256=Windows-1256* • *Modo compatibilidad ASCII:* *0: Occidental* *1: Japonés* *2: Chino* *3: Coreano* *4: Definido por el usuario* *5: Reservado* *6: Europa central* *7: Cirílico* *8: Árabe* *9: Griego* *10: Hebreo* *11: Turco* *12: Nórdico*
Web Client IP address to listen	*Entero largo*	*23*	***Alcance**: todas las máquinas remotas 4D* ***Valores posibles**: ver Web IP address to listen* ***Descripción**: se utiliza para especificar este parámetro para todas las máquinas 4D remotas utilizadas como servidores web (aplicado al servidor web remoto únicamente).*
Web debug log	*Entero largo*	*84*	***Alcance:** servidor web local* ***Nota:** si se reinicia el servidor HTTP, se utiliza un nuevo archivo de historial* ***Descripción:** le permite obtener o definir el estado del archivo de historial de peticiones HTTP del servidor Web 4D. Cuando se activa, este archivo, llamado "**HTTPDebugLog_nn.txt**", se guarda en la carpeta "Logs" de la aplicación (nn es el número de archivo). Es útil para problemas de depuración relacionados con el servidor web. Registra cada petición y cada respuesta en modo raw. La totalidad de las peticiones, encabezados, se registran; opcionalmente, también se pueden registrar partes del cuerpo. Para mayor información sobre archivos HTTPDebugLog, consulte la sección **Anexo E: Descripción de archivos de historial**.* ***Valores**: una de las constantes con el prefijo "wdl" (consulte las descripciones de estas constantes en este tema).* ***Valor por defecto**: 0 (no activado)*
Web HSTS enabled	*Entero largo*	*86*	***Alcance**: 4D local, 4D Server.* ***Descripción**: estado HTTP Strict Transport Security (HSTS). HSTS permite que el servidor Web 4D declare que los navegadores solo deberían interactuar con él a través de conexiones HTTPS seguras. Una vez activado, el servidor web 4D agregará automáticamente información relacionada con HSTS a todos los encabezados de respuesta.Los navegadores registrarán la información HSTS la primera vez que reciban una respuesta del servidor web 4D, luego toda solicitud HTTP futura se transformará automáticamente en solicitudes HTTPS. El tiempo que el navegador almacena esta información se especifica con el selector Web HSTS max age. HSTS requiere que HTTPS esté habilitado en el servidor. HTTP también debe estar habilitado para permitir las conexiones iniciales del cliente.* ***Valores posibles**: 0 (desactivado, predeterminado) o 1 (activado)* ***Nota:** el servidor web 4D debe reiniciarse para que se aplique esta configuración.*

Constante	Tipo	Valor	Comentario
Web HSTS max age	*Entero largo*	*87*	***Alcance**: 4D local, 4D Server* ***Descripción**: especifica el tiempo máximo (en segundos) que HSTS está activo para cada conexión de cliente nuevo. Esta información se almacena en el lado del cliente durante la duración especificada.* ***Valores posibles**: Entero largo (segundos)* ***Valor por defecto**: 63072000 (2 años)* ***Atención:** una vez que HSTS esté habilitado, las conexiones cliente continuarán usando este mecanismo por la duración especificada. Cuando esté probando sus aplicaciones, se recomienda establecer una duración corta para poder cambiar entre los modos de conexión segura y no segura si es necesario.*
Web HTTP compression level	*Entero largo*	*50*	***Alcance**: Servidor web local* ***Descripción**: nivel de compresión para todos los intercambios HTTP comprimidos efectuados para el servidor HTTP de 4D (peticiones cliente o respuestas servidor, Web y servicio web). Este selector permite optimizar los intercambios con un enfoque en la velocidad de ejecución (menor compresión) o la cantidad de compresión (menor velocidad). La elección de un valor depende del tamaño y la naturaleza de los datos intercambiados. Pase de 1 a 9 en el parámetro valor, 1 es la compresión más rápida y 9 la más alta. También puede pasar -1 para obtener un compromiso entre velocidad y tasa de compresión. El nivel de compresión por defecto es 1 (compresión rápida).* ***Valores posibles**: 1 a 9 (1 = más rápido, más comprimido = 9) o -1 = mejor compromiso.*
Web HTTP compression threshold	*Entero largo*	*51*	***Alcance**: Servidor HTTP local* ***Descripción**: en intercambios HTTP optimizados, límite de tamaño de petición por debajo del cual los intercambios no deben comprimirse. Esta opción es útil para evitar pérdidas de tiempo de máquina para comprimir intercambios muy pequeños.* *Pase en valor un tamaño en bytes. Por defecto, el límite de compresión se establece en 1024 bytes.* ***Valores posibles**: todo valor de tipo entero largo. El parámetro valor contiene una tamaño expresado en bytes. Por defecto, el umbral de compresión está definido en 1024 bytes.*
Web HTTP enabled	*Entero largo*	*88*	***Alcance**: 4D local, 4D Server* ***Descripción**: estados para comunicación sobre HTTP.* ***Valores posibles**: 0 (desactivado) o 1 (activado)*
Web HTTP TRACE	*Entero largo*	*85*	***Alcance**: servidor web local* ***Descripción**: le permite activar o desactivar el método HTTP TRACE en el servidor web de 4D. Por razones de seguridad, a partir de 4D v15 R2, por defecto, el servidor web 4D rechaza peticiones TRACE HTTP con un error 405 (ver desactivación de HTTP TRACE). Si es necesario, puede activar el método HTTP TRACE para la sesión pasando esta constante con el valor 1. Cuando se activa esta opción, el servidor web 4D responde a las solicitudes HTTP TRACE con la línea de petición, el encabezado y el cuerpo.* ***Valores posibles**: 0 (desactivado) o 1 (activado)* ***Valor por defecto**: 0 (desactivado)*
Web HTTPS enabled	*Entero largo*	*89*	***Alcance**: 4D local, 4D Server* ***Descripción**: estado para comunicación sobre HTTPS.* ***Valores posibles**: 0 (desactivado) o 1 (activado)*
Web HTTPS port ID	*Entero largo*	*39*	***Alcance**: 4D local, 4D Server* ***Valores posibles**: 0 a 65535* ***Descripción**: número del puerto TCP utilizado por el servidor web de 4D en modo local y de 4D Server para conexiones seguras vía TLS (protocolo HTTPS). El número de puerto HTTPS se define en la página "Web/Configuración" de la caja de diálogo Propiedades de la base.* *Por defecto, el valor es 443 (valor estándar). Puede utilizar las constantes del tema **Números de puerto TCP** para el parámetro valor.*
Web inactive process timeout	*Entero largo*	*78*	***Alcance**: servidor web local* ***Descripción**: permite modificar el timeout del proceso utilizado para la sesión (opción relativa al proceso). Después del timeout, el proceso se elimina en el servidor, se llama al **Método base On Web Close Process** y luego el contexto de la sesión se destruye.* ***Valores**: Entero largo (minutos)* ***Valores por defecto**: 480 minutos (pase 0 para restablecer el valor por defecto)*
Web inactive session timeout	*Entero largo*	*72*	***Alcance**: servidor web local* ***Descripción**: permite modificar la duración de vida de las sesiones inactivas (duración definida en cookie). Al final de este periodo, la cookie de sesión expira y no se envía más al cliente HTTP.* ***Valores**: Entero largo (minutos)* ***Valores por defecto**: 480 minutos (pase 0 para restablecer el valor por defecto)*
Web IP address to listen	*Entero largo*	*16*	***Alcance**: 4D local, 4D Server* ***Descripción**: dirección IP en la que el servidor web debe recibir las peticiones HTTP con 4D en modo local y 4D Server. Por defecto, ninguna dirección se especifica. Este parámetro se define en las Propiedades de la base. Este selector es útil en el caso de los servidores web 4D compilados y fusionados con 4D Desktop (no hay acceso al modo Diseño).* ***Valores posibles**: dirección IP en forma de cadena. Ambos formatos cadena IPv6 (por ejemplo, "2001:0db8:0000:0000:0000:ff00:0042:8329") y los formatos cadena IPv4 (por ejemplo, "123.45.67.89") son compatibles.* ***Nota:** por compatibilidad, las direcciones IPv4 expresadas en como longitudes hexadecimales (obsoletas) aún son compatibles.*

Constante	Tipo	Valor	Comentario
Web keep session	*Entero largo*	*70*	***Alcance**: servidor web local* ***Descripción**: permite activar o desactivar el modo de gestión de las sesiones (descrito en la sección **Gestión de las sesiones web**).* ***Valores**: 1 (activar modo) ó 0 (desactivar modo)* ***Valor por defecto**: 1 para bases creadas en la versión 13, 0 para bases convertidas. Note que este modo activa igualmente el mecanismo de reutilización de los contextos temporales en modo remoto. Para mayor información sobre este mecanismo, consulte la descripción de esta opción en la sección **Parámetros del servidor web**.*
Web log recording	*Entero largo*	*29*	***Alcance**: 4D local 4D Server* ***Descripción**: inicia o detiene el registro de peticiones solicitudes Web recibida por el servidor web de 4D en modo local o 4D Server. Por defecto, el valor es 0 (no hay registro de peticiones).* *El historial de las peticiones web se guarda en un archivo texto llamado "logweb.txt" que se ubica automáticamente en la carpeta Logs de la base, junto al archivo de estructura. El formato de este fichero es determinado por el valor que se pase. Para más información sobre los diferentes formatos de historial de las peticiones, consulte la sección [#title id= "2833"/]. La activación de este archivo también se puede definir en la página "Web/Avanzado" de las Preferencias de 4D.* ***Valores posibles**: 0 = No guardar (por defecto), 1 = Registrar en formato CLF, 2 = Registrar en formato DLF, 3 = Registrar en formato DLF, 4 = Guardar en formato WLF.* ***Atención**: los formatos 3 y 4 formatos son formatos personalizados, los contenidos deben ser definidos de antemano en las Preferencias de la aplicación, página "Web/Formato del historial". Si usted utiliza uno de estos formatos sin que sus campos hayan sido seleccionados, el archivo de las peticiones no se generará.*
Web max concurrent processes	*Entero largo*	*18*	***Alcance**: 4D local, 4D Server* ***Descripción**: límite estrictamente superior del número de procesos web de todo tipo aceptados por el servidor web con 4D en modo local y 4D Server. Cuando se alcanza el número límite (menos uno), 4D no crea un nuevo proceso y devuelve el mensaje "Servidor no disponible" (estado HTTP 503 - Servicio no disponible) a toda nueva petición.* *Este parámetro previene la saturación del servidor web de 4D que puede ocurrir durante un envío masivo de peticiones o de una demanda excesiva de creación de contextos. También puede definirse en la caja de diálogo de la Propiedades de la base.* *En teoría, el número máximo de procesos web es el resultado de dividir la memoria disponible / tamaño de la pila de un proceso web. Otra solución es ver la información sobre los procesos web que se muestra en el Explorador de ejecución: se indican el número actual de procesos web y el número máximo alcanzado desde el inicio del servidor web.* ***Valores**: todo valor entre 10 y 32 000. El valor por defecto es 100.*
Web max sessions	*Entero largo*	*71*	***Alcance**: servidor web local* ***Descripción**: permite limitar el número de sesiones simultáneas. Cuando se alcanza el número definido, la sesión más antigua se cierra (y se llama al **Método base On Web Close Process**) si el servidor web necesita crear una nueva.* ***Valores posibles:** Entero largo. El número de sesiones simultáneas no puede superar el número total de procesos web (opción Web Max Concurrent Processes, 100 por defecto)* ***Valores por defecto**: 100 (pase 0 para restablecer el valor por defecto)*
Web maximum requests size	*Entero largo*	*27*	***Alcance**: 4D local, 4D Server* ***Descripción**: tamaño máximo (en bytes) de las peticiones HTTP entrantes (POST) que el servidor web está autorizado a tratar. Por defecto, el valor predeterminado es 2 000 000, es decir, un poco menos de 2 MB. El valor máximo (2 147 483 648) significa en la práctica que ningún límite se establece.* *Esta configuración evita la saturación del servidor web, causadas por peticiones entrantes muy grandes. Cuando una petición llega al límite, el servidor web de 4D la rechaza.* ***Valores posibles**: 500 000 a 2 147 483 648.*
Web port ID	*Entero largo*	*15*	***Alcance**: 4D en modo local y 4D Server.* ***Descripción**: establece u obtiene el número del puerto TCP utilizado por el servidor web 4D con 4D en modo local y 4D Server. Por defecto, el valor es 80. El número de puerto TCP se define en la página "Web/Configuración" de la caja de diálogo Propiedades de la base. Puede utilizar una de las constantes del tema **Números de puerto TCP** para el parámetro valor. Este selector es útil en el marco de servidores web 4D que se compilan y fusionan utilizando 4D de escritorio (sin acceso al entorno Diseño).* ***Valores posibles**: para obtener más información sobre el número de puerto TCP, consulte la sección **Parámetros del servidor web**.* ***Valor por defecto**: 80*
Web session cookie domain	*Entero largo*	*81*	***Alcance**: Servidor web local* ***Descripción**: define u obtiene el valor del campo "dominio" de la cookie de sesión. Este selector (así como el selector 82) es útil para controlar el alcance de las cookies de sesión: si configura, por ejemplo, el valor "/*.4d.fr" para este selector, el cliente sólo enviará una cookie cuando la petición se dirige al dominio ".4d.fr", que excluye los servidores que alojan los datos estáticos externos.* ***Valores posibles**: Texto*
Web session cookie name	*Entero largo*	*73*	***Alcance**: servidor web local* ***Descripción**: permite definir el nombre de la cookie utilizada para almacenar el ID de la sesión.* ***Valores**: Texto* ***Valores por defecto**: "4DSID" (pase una cadena vacía para restablecer el valor por defecto)*

Constante	Tipo	Valor	Comentario
Web session cookie path	*Entero largo*	*82*	***Alcance**: servidor web local* ***Descripción**: define u obtiene el valor del campo "path" de la cookie de sesión. Este selector (así como elselector 81) es útil para controlar el alcance de las cookies de sesión: si configura, por ejemplo, el valor "/4DACTION" para este selector, el cliente deberá enviar sólo una cookie para peticiones dinámicas que comiencen con 4DACTION , y no para las imágenes, páginas estáticas, etc.* ***Valores posibles**: Texto*
Web session enable IP address validation	*Entero largo*	*83*	***Alcance**: servidorWeb Local* ***Descripción**: Activa o desactiva la validación de las direcciones IP para las cookies de sesión. Por razones de seguridad, por defecto, el servidor web de 4D verifica la dirección IP de cada solicitud que contiene una cookie de sesión y la rechaza si esta dirección no coincide con la dirección IP utilizada para crear la cookie. En algunas aplicaciones específicas, es posible que desee desactivar esta validación y aceptar las cookies de sesión, incluso cuando sus direcciones IP no coincidan. Por ejemplo, cuando los dispositivos móviles cambian entre redes WiFi y 3G/4G, su dirección IP cambia. En este caso, debe pasar 0 en esta opción para permitir que los clientes puedan seguir utilizando sus sesiones web incluso cuando las direcciones IP cambien. Tenga en cuenta que esta configuración reduce el nivel de seguridad de la aplicación.* *Cuando se modifica, esta configuración es efectiva inmediatamente (no es necesario reiniciar el servidor HTTP).* ***Valores posibles**: 0 (desactivado) o 1 (activado)* ***Valor por defecto**: 1 (las direcciones IP son verificadas)*

Cuando utiliza el selector Web debug log, puede recibir una de las siguientes constantes en el parámetro valor:

Constante	Tipo	Valor	Comentario
wdl disable	*Entero largo*	*0*	*El archivo de historial de peticiones HTTP Web está desactivado*
wdl enable with all body parts	*Entero largo*	*7*	*El archivo de historial de peticiones HTTP Web está activado con el cuerpo de la respuesta y la respuesta*
wdl enable with request body	*Entero largo*	*5*	*El archivo de historial de peticiones HTTP Web está activado con el cuerpo de la respuesta únicamente*
wdl enable with response body	*Entero largo*	*3*	*El archivo de historial de peticiones HTTP Web está activado con el cuerpo de la respuesta únicamente*
wdl enable without body	*Entero largo*	*1*	*El archivo de historial de peticiones HTTP Web está desactivado sin el cuerpo (el tamaño del cuerpo se entrega en este caso)*

WEB Get server info

WEB Get server info {(conCaché)} -> Resultado

Parámetro	*Tipo*		*Descripción*
conCaché	*Booleano*	→	*True para devolver la descripción de la caché Web. De lo contrario (por defecto), no se devuelve la descripción de la caché.*
Resultado	*Objeto*	↩	*Información sobre el servidor Web en ejecución y el servidor SOAP*

Descripción

*El comando **WEB Get server info** devuelve un objeto que contiene información del tiempo de ejecución detallada en la sesión actual del servidor web 4D. La información devuelta incluye el servidor SOAP.*

***Nota:** este comando devuelve información de tiempo de ejecución, es decir, parámetros reales utilizados por el servidor web. Estos parámetros pueden diferir de los devueltos por el comando **WEB GET OPTION** ya que dependen de la configuración del sistema, los recursos disponibles, etc.*

De forma predeterminada, el comando no devuelve la propiedad "caché", ya que puede ser muy grande. Sin embargo, si desea conocer el contenido de la caché, pase True en el parámetro opcional conCache.

El objeto devuelto contiene las siguientes propiedades (los nombres de propiedad son sensibles a las mayúsculas y minúsculas):

Nombre de la propiedad			Tipo de valor	Descripción
started			Booleano	true si se inicia el servidor http, de lo contrario false
uptime			Número	Tiempo transcurrido desde el último inicio del servidor http
httpRequestCount			Número	Número de visitas HTTP recibidas por el servidor desde que se inició
startMode			Cadena	"automático" si está activada la opción "Iniciar servidor web al arrancar", de lo contrario "manual".
SOAPServerStarted			Booleano	true si se inicia el servidor SOAP, en caso contrario false
cache			Objeto	Esta propiedad sólo se incluye si el valor de la propiedad del parámetro cacheInfo es True. Describe el contenido de la caché del servidor web (ver **Propiedad cache** a continuación)
security			Objeto	Estado actual de las distintas opciones de seguridad
	properties		Objeto	Estado actual de las diversas propiedades de seguridad
		cipherSuite	Cadena	Lista de cifras utilizada por 4D para el protocolo seguro (corresponde al parámetro de la base SSL cipher list)
		HTTPEnabled	Booleano	True si HTTP está activado
		HTTPSEnabled	Booleano	True si HTTPS está activado
		HSTSEnabled	Booleano	True si HSTS está activado en el servidor
		HSTSMaxAge	Número	Edad máxima (en segundos) para HSTS. El valor por defecto es 2 años (63,072,000 segundos).
		minTLSVersion	Cadena	Versión TLS mínima aceptada para las conexiones (corresponde al parámetro de base de datos Min TLS version)
		openSSLVersion	Cadena	Versión de la librería OpenSSL utilizada
		perfectForwardSecrecy	Booleano	True si PFS está disponible en el servidor, de lo contrario False
options			Objeto	Estado actual de varias opciones del servidor web estándar
	properties		Objeto	Estado actual de varias propiedades de opción del servidor web estándar
		webCharacterSet	Cadena	Nombre de conjunto de caracteres (corresponde a la opción web Web character set)
		webHTTPCompressionLevel	Número	Nivel de compresión para los intercambios HTTP comprimidos (corresponde a la opción web Web HTTP compression level)
		webHTTPCompressionThreshold	Number	Valor de compresión (corresponde a la opción web Web HTTP compression threshold)
		webHTTPSPortID	Number	Número de puerto TCP utilizado por el servidor Web para conexiones seguras (corresponde a la opción web Web HTTPS port ID)
		webInactiveProcessTimeout	Número	Duración de vida de los procesos de sesión inactivos (corresponde a la opción web Web inactive process timeout)
		webInactiveSessionTimeout	Número	Duración de la vida de las sesiones inactivas (corresponde a la opción web [#cst id="844884"/])
		webIPAddressToListen	Colección	La dirección IP en el "formato" definido en el que el servidor web recibe las solicitudes http (corresponde a la opción web Web IP address to listen)
		webMaxConcurrentProcesses	Número	Número máximo de procesos web simultáneos (corresponde a la opción web [#cst id="843981"/])
		webPortID	Número	Puerto TCP utilizado por el servidor Web (corresponde a la opción web Web port ID)

***4D Server**: el comando devuelve información sobre el servidor web local. Si desea monitorear el servidor web 4D Server desde un 4D remoto, debe aplicar la propiedad "Ejecutar en el servidor" al método.*

Propiedad cache

*Si pasa **true** en el parámetro conCache, el comando devuelve la propiedad objeto "cache" con el siguiente contenido:*

Nombre de la propiedad	Tipo de valor	Descripción
cacheUsage	Número	Tasa de uso de la caché
numOfLoads	Número	Número de objetos cargados
currentSize	Número	Tamaño actual de la caché
maxSize	Número	Tamaño máximo de la caché
objectMaxSize	Número	Tamaño máximo de objetos cargables en la caché
enabled	Booleano	true si la caché del servidor web está activada
nbCachedObjects	Número	Número de objetos en la caché
cachedObjects	Colección	Colección de objetos en la caché. Cada objeto en caché es definido por diferentes propiedades (url, mimeType, expirationType, lastModified, etc.)

Ejemplo

Después de ejecutar el siguiente código:

```
$webServerInfo:=WEB Get server info(True)
```

... $webServerInfo contendrá por ejemplo:


```
{ "started": true, "uptime": 40, "SOAPServerStarted": true, "startMode": "manual", "httpRequestCount": 0, "options": { "webCharacterSet": "UTF-8", "webHTTPCompressionLevel": 1, "webHTTPCompressionThreshold": 1024, "webHTTPSPortID": 443, "webIPAddressToListen": ["192.168.xxx.xxx"], "webInactiveProcessTimeout": 28800, "webInactiveSessionTimeout": 28800, "webMaxConcurrentProcesses": 100, "webPortID": 80 }, "security": { "HTTPEnabled": true, "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256:...:CAMELLIA128-SHA", "openSSLVersion": "OpenSSL 1.0.2h 3 May 2016", "perfectForwardSecrecy": true, "minTLSVersion": "1.2" }, "cache": { "cacheUsage": 1, "numOfLoads": 24, "currentSize": 154219, "maxSize": 10485760, "objectMaxSize": 524288, "enabled": true, "nbCachedObjects": 23, "cachedObjects": [ {...},{...} ] } }
```

WEB GET SESSION EXPIRATION

WEB GET SESSION EXPIRATION (idSesion ; fechaVenc ; horaVenc)

Parámetro	***Tipo***		***Descripción***
idSesion	*Texto*	➡	*UUID de sesión*
fechaVenc	*Fecha*	⬅	*Fecha de vencimiento de la cookie*
horaVenc	*Hora*	⬅	*Fecha de vencimiento de la cookie*

Descripción

*El comando **WEB GET SESSION EXPIRATION** devuelve la información relativa al vencimiento de la cookie de la sesión cuyo UUID se pasó en idSesion.*

El parámetro fechaVenc recibe la fecha de vencimiento y el parámetro horaVenc recibe la hora de vencimiento de la cookie.

***Nota:** cada vez que se envía una petición web, la fecha y hora de vencimiento de la cookie se reinician en un valor correspondiente a la hora de la petición+el valor de la opción Web Inactive session timeout. Por ejemplo:*
Primera petición, lunes a la 1:00
-> envía una cookie 4DSID xxxyyy vence I+24h = martes 01:00
Segunda petición, lunes a la 1:10
-> envía una cookie 4DSID xxxyyy vence I+24h = martes 01:10
Tercera petición, martes a las 4:00: cookie vencida
-> envía una cookie 4DSID aaabbb vence I+24h = miércoles 01:00

WEB Get session process count

WEB Get session process count (idSesion) -> Resultado

Parámetro	Tipo		Descripción
idSesion	Texto	➡	UUID de sesión
Resultado	Entero largo	⮌	Número de procesos vinculados a la sesión

Descripción

*El comando **WEB Get session process count** devuelve el número de procesos vinculados a la sesión cuyo UUID se pasó en idSesion.*

*Este comando se añade en el contexto de la funcionalidad **Gestión de sesiones 4D Mobile** introducida en 4D v15 R4. Está diseñado principalmente para contar el número de procesos ejecutados por una sesión 4D Mobile.*

- *Para una sesión 4D Mobile, este comando devuelve el número real de procesos. Una sesión 4D Mobile puede ejecutar varios procesos.*
- *Para una sesión Web regular, este comando siempre devuelve 1 (una sesión Web = un proceso).*

Ejemplo

Usted desea almacenar información sobre la sesión 4D Mobile actual en arrays:

```
C_TEXT($sessionID)
C_LONGINT($count)
C_DATE($expDate)
C_TIME($expTime)

$sessionID:=WEB Get Current Session ID
$count:=WEB Get session process count($sessionID)
WEB GET SESSION EXPIRATION($sessionID;$expDate;$expTime)

APPEND TO ARRAY($aTimestamp;String(Current date)+" "+String(Current time))
APPEND TO ARRAY($aSessionUID;$sessionID)
APPEND TO ARRAY($aNbProcesses;$count)
APPEND TO ARRAY($aExpirationDate;$expDate)
APPEND TO ARRAY($aExpirationTime;$expTime)
```

WEB GET STATISTICS

WEB GET STATISTICS (paginas ; hits ; uso)

Parámetro	Tipo		Descripción
paginas	Array texto	⇐	Nombres de las páginas más consultadas
hits	Array entero largo	⇐	Número de hits para cada página
uso	Entero largo	⇐	Porcentaje de la caché utilizado

Descripción

*El comando **WEB GET STATISTICS** permite obtener la información sobre las páginas más consultadas, cargadas en la caché del servidor web. Por lo tanto, estas estadísticas conciernen únicamente las páginas estáticas, las imágenes GIF, las imágenes JPEG <100 KB y las hojas de estilo (.css).*

***Nota:** para mayor información sobre el parámetro de la caché del servidor web 4D, consulte la sección **Parámetros del servidor web**.*

El comando llena el array de texto páginas con los nombres de las páginas más consultadas. El array entero largo hits recibe el número de "hits" por cada página. La variable numérica uso recibe el porcentaje de la caché web utilizada por cada página.

Ejemplo

Asumamos que quiere generar una página semidinámica que muestre las estadísticas de utilización de la caché web. Para esto, en una página HTML estática llamada "stats.shtm" (las páginas con sufijo .shtm son analizadas automáticamente por el servidor web), ponga la etiqueta <!--4DSCRIPT/STATS--> así como las referencias a las variables vPages y vUsage, por ejemplo:

```
<html> <head><title>4D Web Stats</title></head> <!--#4DSCRIPT/STATS--> <body> <strong>Porcentaje de caché utilizada: </strong>
<!--#4DTEXT vUsage--> <hr> <strong>Páginas más consultadas: </strong> <!--#4DHTML vPages--> </body> </html>
```

*En el método de proyecto **STATS**, escriba el siguiente código:*

```
C_TEXT($1)
C_TEXT(vPages)
ARRAY TEXT(paginas;0)
ARRAY LONGINT(hits;0)
C_LONGINT(vUsage)
WEB CACHE STATISTICS(paginas;hits;vUsage)

For($i;1;Size of array(paginas))
   \\Para cada página presente en la caché
   vPaginas:=paginas{$i}+" "+String(hits{$i})+"
   " \\Inserte el nombre de la página y el código HTML
End for
```

*Puede enviar la página "stats.shtm" utilizando un enlace URL o utilizando el comando **WEB SEND FILE**.*

WEB GET VARIABLES

WEB GET VARIABLES (arrayNoms ; arrayValores)

Parámetro	Tipo		Descripción
arrayNoms	Array texto	⬅	Nombres de las variables del formulario web
arrayValores	Array texto	⬅	Valores de las variables del formulario web

Descripción

*El comando **WEB GET VARIABLES** llena los arrays texto arrayNoms y arrayValores con los nombres y los valores de las variables contenidas en el formulario web "enviado" al servidor web.*

Este comando obtiene el valor de todas las variables que pueden incluirse en páginas HTML: áreas de entrada, botones, casillas de selección, botones de radio, menús pop up, listas de opciones...

***Nota:** en el caso de las casillas de selección, el nombre de la variable y su valor sólo se devuelven si la casilla de selección ha sido seleccionada.*

Este comando es valido sin importar el tipo de URL o de formulario (método POST o GET) enviado al servidor web.

*Este comando puede llamarse, si es necesario, en el **Método de base On Web Connection** o en cualquier otro método 4D que resulte del envío de un formulario.*

Acerca de los formularios Web y sus acciones asociadas

Un formulario contiene áreas de entrada (áreas de texto, botones, casillas de selección), cada una con un nombre.
Cuando un formulario se "envía" al servidor web (una petición se envía al servidor web), la petición contiene, entre otros, la lista de áreas de entrada y sus valores respectivos.
Hay dos métodos para enviar un formulario (ambos pueden utilizarse indiferentemente con 4D):

- *POST, generalmente utilizado para añadir datos en el servidor web, a una base de datos,*
- *GET, generalmente utilizado para la petición del servidor web, datos que provienen de una base de de datos.*

Ejemplo

Un formulario contiene dos campos, vNombre y vCiudad que reciben los valores "ROBERTO" y "PARIS". La acción asociada al formulario es "/4DACTION/WEBFORM".

- *Si el método de formulario es POST (el utilizado con más frecuencia), los datos introducidos no serán visible en el URL (http://127.0.0.1/4DACTION/WEBFORM).*
- *Si el método de formulario es GET, los datos serán visibles en el URL (http://127.0.0.1/4DACTION/WEBFORM?vNOMBRE=ROBERTO&vCIUDAD=PARIS).*

El método WEBFORM puede ser de esta forma:

```
ARRAY TEXT($anombres;0)
ARRAY TEXT($avalores;0)
WEB GET VARIABLES($anombres;$avalores)
```

El resultado será:

```
$anombres{1}="vNOMBRE"
$anombres{2}="vCIUDAD"
$avalores{1}="ROBERTO"
$avalores{2}="PARIS"
```

La variable vNOMBRE contiene ROBERTO y vCIUDAD contiene PARIS.

WEB Is secured connection

WEB Is secured connection -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*True = la conexión web es segura. False = la conexión web no es segura.*

Descripción

*El comando **WEB Is secured connection** devuelve un booleano indicando si la conexión al servidor web 4D se efectúo en modo seguro por medio de TSL/SSL (la petición comienza por "https:" en lugar de "http:").*

- *Si la conexión se realiza en TLS o SSL, la función devuelve True.*
- *Si la conexión se realiza en modo no seguro, la función devuelve False.*

***Nota:** para mayor información sobre el protocolo SSL, consulte la sección **Utilizar el protocolo TLS (HTTPS)**.*
Este comando permite, por ejemplo, rechazar los intentos de conexión en modo no seguro.

WEB Is server running

WEB Is server running -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano	➲	True si el servidor Web está corriendo, de lo contrario False

Descripción

*El comando **WEB Is server running** devuelve **True** si el servidor web integrado 4D está corriendo y **False** si el servidor web está apagado.*

Este comando ofrece el estado de funcionamiento del servidor web en el que se ejecuta:

Contexto de ejecución	Devuelve el estado de
4D monopuesto	Servidor web 4D local
4D Server	Servidor web 4D Server
4D modo remoto (proceso local)	Servidor web 4D local
4D modo remoto (procedimiento almacenado en 4D Server)	Servidor web 4D Server
4D modo remoto (procedimiento almacenado en otro 4D remoto)	Servidor web 4D remoto

Ejemplo

Usted quiere verificar que el servidor web se está ejecutando:

```
If (WEB Is server running)
   ... //hacer las acciones apropiadas
End if
```

WEB SEND BLOB

WEB SEND BLOB (BLOB ; tipo)

Parámetro	Tipo		Descripción
BLOB	BLOB	➡	BLOB a enviar al navegador
tipo	Cadena	➡	Tipo de datos del BLOB

Descripción

*El comando **WEB SEND BLOB** permite enviar el BLOB blob al navegador.*
El tipo de datos contenidos en el BLOB es indicado por tipo. Este parámetro puede contener los siguientes valores:

- *tipo = **Cadena vacía** (""): en este caso, no necesita dar más información en el BLOB. El navegador intentará interpretar los contenidos del BLOB.*
- *tipo = **Extensión de archivo** (Ejemplo: ".HTM", ".GIF", ".JPEG", etc.): en este caso, especifique el navegador, por intermedio de su extensión, el tipo MIME de los datos contenidos en el BLOB. El BLOB será interpretado de acuerdo a su extensión. Sin embargo, la extensión debe ser estándar para que el navegador pueda interpretarla correctamente.*
- *tipo = **Mime/Tipo** (Ejemplo: "text/html", "image/tiff", etc.): en este caso, especifique directamente al navegador el tipo MIME de los datos contenidos en el BLOB. Esta solución ofrece más libertad. Además, los tipos estándar, puede pasar un tipo MIME personalizado para enviar los documentos propietarios en Intranet. Para hacerlo, sólo necesita configurar los navegadores con el fin de reconocer el tipo enviado y ejecutar la aplicación correspondiente. El valor a pasar en el parámetro tipo es, en este caso, "application/x-[NombreTipo]". En los navegadores de los equipos clientes, usted referencia este tipo y lo asocia a la acción "Launch the application". El comando **WEB SEND BLOB** permite entonces enviar documentos de todo tipo, los clientes Intranet abren automáticamente la aplicación asociada.*

  ***Nota:** para mayor información sobre los tipos MIME, consulte la página: http://www.iana.org/assignments/media-types.*

Esta es una lista de los tipos MIME más comunes:

Extensión	Mime/Tipo
.htm	text/html
.html	text/html
.shtml	text/html
.shtm	text/html
.css	text/css
.pdf	application/pdf
.rtf	application/rtf
.ps	application/postscript
.eps	application/postscript
.hqx	application/mac-binhex40
.js	application/javascript
.json	application/json
.txt	text/plain
.text	text/plain
.gif	image/gif
.jpg	image/jpeg
.jpeg	image/jpeg
.jpe	image/jpeg
.jfif	image/jpeg
.pic	image/pict
.pict	image/pict
.tif	image/tiff
.tiff	image/tiff
.mpeg	video/mpeg
.mpg	video/mpeg
.mov	video/quicktime
.moov	video/quicktime
.aif	audio/aiff
.aiff	audio/aiff
.wav	audio/wav
.ram	audio/x-pn-realaudio
.sit	application/x-stuffit
.bin	application/x-stuffit
.xml	application/xml
.z	application/x-zip
.zip	application/x-zip
.gz	application/x-gzip
.tar	application/x-tar

Nota: *la lista de tipos MIME soportada por el servidor 4D HTTP se guarda en el archivo "MimeTypes.xml" que se encuentra en la siguiente carpta de la aplicación 4D: [Contents]\Native components\HTTPServer.bundle\Contents\Resources.*

Las referencias a las variables 4D y etiquetas de tipo 4DSCRIPT en la página siempre se analizan.

Ejemplo

Consulte el ejemplo de la rutina ***PICTURE TO BLOB****.*

WEB SEND FILE

WEB SEND FILE (archivohtml)

Parámetro	Tipo		Descripción
archivohtml	Cadena	➡	Ruta de acceso HTML al archivo HTML o cadena vacía para terminar SEND HTML FILE

Descripción

*El comando **WEB SEND FILE** envía al navegador web la página HTML o el archivo web almacenado en el documento cuya ruta se pasa en archivohtml.*

Por defecto, 4D busca el documento HTML al interior de la carpeta raíz, definida en las Propiedades de la base.
Este comando acepta como parámetro una ruta de acceso expresada en sintaxis Poxis (nombres de directorios o de carpetas separados por una barra oblicua "/") o en sintaxis sistema.
*Si pasa una ruta de acceso inválida, se genera un error asociado a la gestión de los archivos de su sistema operativo. Puede interceptar este error utilizando un método instalado por el comando **ON ERR CALL**. Si el método muestra una caja de diálogo de alerta o de mensaje, aparecerá en el equipo del navegador.*
*Una vez se ejecuta **WEB SEND FILE**, la variable sistema OK se actualiza: si el archivo a enviar existe y si el timeout no ha pasado, OK toma el valor 1. De lo contrario, toma el valor 0.*

***Nota:** si llama **WEB SEND FILE** desde un proceso que no es un proceso web, el comando no hace nada. No se devuelve ningún error y la llamada simplemente se ignora.*

Las referencias a las variables 4D y a las etiquetas de tipo 4DSCRIPT en la página siempre se analizan cuando el tipo de documento lo permite (documento basado en texto).

Ejemplo

La carpeta raíz HTML de la base es la carpeta WebDocs. Contiene los siguientes elementos:

.

```
.\WebDocs\HTM\MiPagina.HTM
```

El envío de la página web "MiPagina.HTM" debe efectuarse de la siguiente forma:

```
WEB SEND FILE("HTM/MiPagina.HTM")
```

Variables y conjuntos del sistema

Si el archivo a enviar existe y si el timeout no ha pasado, OK toma el valor 1. De lo contrario, toma el valor 0.

WEB SEND HTTP REDIRECT

WEB SEND HTTP REDIRECT (url {; *})

Parámetro	Tipo		Descripción
url	Cadena	➡	Nuevo URL
*	Operador	➡	Si se especifica = el URL no está traducido, Si se omite = el URL está traducido

Descripción

*El comando **WEB SEND HTTP REDIRECT** permite transformar una URL en otra.*

*El parámetro url contiene el nuevo URL que permite redirigir la petición. Si este parámetro es un url a un archivo, debe contener la referencia a este archivo, por ejemplo: **WEB SEND HTTP REDIRECT** ("/MiPagina.HTM").*

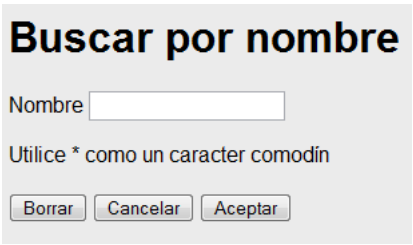
*Este comando prevalece sobre los comandos de envío de datos (**WEB SEND FILE**, **WEB SEND BLOB**, etc.) que puedan estar en el mismo método.*

Este comando también permite redirigir una petición a otro servidor web.

*4D codifica automáticamente los caracteres especiales del URL. Si pasa el carácter *, 4D no los traducirá.*
*Note que el estado de la petición enviada por este comando es **302: Moved Temporarily**. Si necesita una redirección permanente (status 301), puede fijar el campo HTTP X-STATUS: 301 en el encabezado de la respuesta.*

Ejemplo

Puede utilizar este comando para efectuar, con la ayuda de páginas estáticas, búsquedas personalizadas en 4D. Imagine que coloca los siguientes elementos en una página HTML estática:



***Nota:** la acción POST "/4dcgi/rech" se ha asociado al área de texto y a los botones **Aceptar** y **Cancelar**.*
*En el **Método base On Web Connection**, inserte el siguiente código:*

```
Case of
   :($1="/4dcgi/rech")  //Cuando 4D recibe este URL
  //Si el botón Aceptar ha sido utilizado y el campo 'nombre' contiene un valor
      If((bOK="OK")  &  (nombre#""))
  //Cambie el URL para ejecutar el código de la búsqueda,
  //ubicado más adelante en el mismo método
         WEB SEND HTTP REDIRECT("/4dcgi/rech?"+name)
      Else
  //Si no volver a la página de inicio
         WEB SEND HTTP REDIRECT("/page1.htm")
      End if
      ...
   :($1="/4dcgi/rech?@")  //Si el URL ha sido redirigido
      ...  //Coloque el código de búsqueda aquí
End case
```

WEB SEND RAW DATA

WEB SEND RAW DATA (datos {; *})

Parámetro	Tipo		Descripción
datos	BLOB	➡	Datos HTTP a enviar
*	Operador	➡	Envío en trozos (chunked)

Descripción

*El comando **WEB SEND RAW DATA** permite al servidor web 4D enviar datos HTTP "brutos", los cuales pueden estar en trozos.*

El parámetro datos contiene las dos partes estándar de una respuesta HTTP, es decir el encabezado y el cuerpo. Los datos son enviados sin formato previo por el servidor. Sin embargo, 4D efectúa un control básico sobre el encabezado y el cuerpo de la respuesta con el fin de asegurarse de que sean válidos:

• Si el encabezado está incompleto o no cumple con las especificaciones del protocolo HTTP, 4D lo modifica como corresponde.
• Si la petición HTTP está incompleta, 4D añade la información faltante. Si por ejemplo quiere efectuar una redirección, debe escribir:

```
HTTP/1.1 302    Location: http://...
```

Si sólo pasa:

```
Location: http://...
```

4D completará la petición añadiendo HTTP/1.1 302.

*El parámetro opcional * permite especificar que la respuesta se enviará "troceada". El corte de las respuestas puede ser útil cuando el servidor envía una respuesta sin conocer su longitud total (si, por ejemplo, la respuesta todavía no ha sido generada). Todos los navegadores compatibles HTTP/1.1-aceptan las respuestas troceadas.*

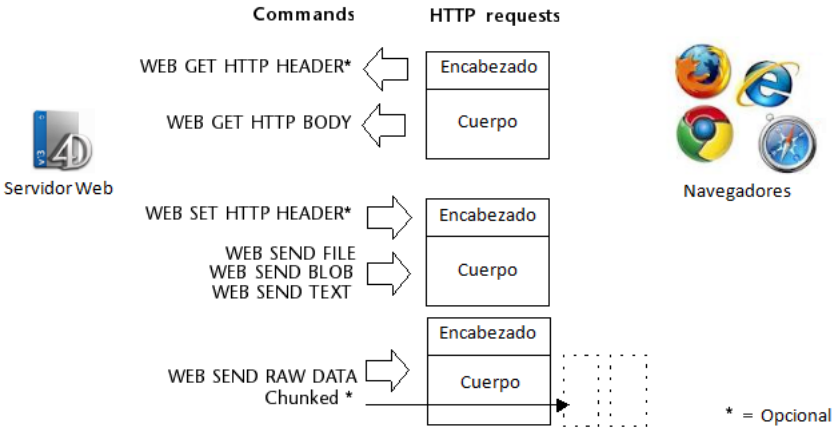
*Si pasa el parámetro *, el servidor web incluirá automáticamente el campo transfer-encoding: chunked en el encabezado de la respuesta, si es necesario (puede manejar manualmente el encabezado de la respuesta, si así lo prefiere). El resto de la respuesta también será formateada para respetar la sintaxis de la opción chunked. Las respuestas troceadas contienen un solo encabezado y un número indefinido de cuerpos.*

*Todas las instrucciones **WEB SEND RAW DATA** que sigan la ejecución de **WEB SEND RAW DATA**(data;*) dentro del mismo método serán consideradas como parte de la respuesta (sin importar si contienen el parámetro *). El servidor coloca fin al envío troceado cuando termina la ejecución del método.*

***Nota:** si el cliente web no soporta el protocolo HTTP/1.1, 4D convertirá automáticamente la respuesta al formato compatible HTTP/1.0 (el envío no troceará). Sin embargo, en este caso, el resultado puede no corresponder a sus deseos. Por lo tanto se recomienda probar si el navegador web soporta HTTP/1.1 y enviar una respuesta apropiada. Para hacer esto, puede utilizar un método de este tipo:*

```
C_BOOLEAN($0)
ARRAY TEXT(arCampos;0)
ARRAY TEXT(arValores;0)
WEB GET HTTP HEADER(arCampos;arValores)
$0:=False
If(Size of array(arValores)>=3)
   If(Position("HTTP/1.1";arValores{3})>0)
      $0:=True  ` El navegador soporta HTTP/1.1; devuelve True en $0
   End if
End if
```

*Combinado con el comando **WEB GET HTTP BODY** y con los otros comandos del tema "Servidor web", este comando completa el rango de herramientas disponibles para los desarrolladores 4D para tratar de manera completamente personalizada las conexiones HTTP entrantes y salientes. Estas herramientas se presentan en el siguiente diagrama:*



Ejemplo

*Este ejemplo ilustra el uso de la opción chunked con el comando **WEB SEND RAW DATA**. Los datos (una secuencia de números) se envían en 100 trozos generados rápidamente un bucle. Recuerde que el encabezado de la respuesta no está definido explícitamente: el comando lo enviará automáticamente e insertará el campo transfer-encoding: chunked en él si el parámetro * se utiliza.*

```
C_LONGINT($cpt)
C_BLOB($mi_blob)
C_TEXT($salida)

For($cpt;1;100)
   $salida:="["+String($cpt)+"]"
   TEXT TO BLOB($salida;$mi_blob;UTF8 text without length)
   WEB SEND RAW DATA($mi_blob;*)
End for
```

WEB SEND TEXT

WEB SEND TEXT (textoHTML {; tipo})

Parámetro	Tipo		Descripción
textoHTML	Texto	➡	Campo o variable de tipo texto con formato HTML a enviar al navegador web
tipo	Texto	➡	True = Ir al modo contextual False o si se omite = Permanecer en el modo actual

Descripción

*El comando **WEB SEND TEXT** envia directamente los datos de texto con formato HTML.*

El parámetro textoHTML contiene los datos a enviar. Como 4D no efectúa ningún control sobre el contenido de este parámetro, asegúrese de que la codificación HTML sea correcta.

Las eventuales referencias a las variables 4D y etiquetas de tipo 4DSCRIPT en el texto siempre se analizan.

*Por defecto, si omite el parámetro tipo, 4D asume que los datos enviados son de tipo "text/html". El comando equivale exactamente al envío de un BLOB de tipo "text/html" utilizando el comando **WEB SEND BLOB**.*

*También puede utilizar el parámetro tipo para especificar el tipo MIME del texto a enviar. Para mayor información sobre los tipos MIME soportados, consulte la descripción del comando **WEB SEND BLOB**.*

Ejemplo

El siguiente método:

```
TEXT TO BLOB("<html><head></head><body>"+String(Current time)+"</body></html>";$blob;UTF8 Text without length)
WEB SEND BLOB($blob;"text/html")
```

... puede reemplazarse por una sola línea:

```
WEB SEND TEXT("<html><head></head><body>"+String(Current time)+"</body></html>")
```

WEB SET HOME PAGE

WEB SET HOME PAGE (pagInicio)

Parámetro	Tipo		Descripción
pagInicio	Cadena	➡	Nombre de la página o ruta de acceso HTML a la página o "" para no enviar la página de inicio personalizada

Descripción

*El comando **WEB SET HOME PAGE** permite modificar la página de inicio personalizada para el proceso web actual.*

La página definida está asociada al proceso web, por lo tanto usted puede definir diferentes páginas de inicio dependiendo, por ejemplo, del usuario que esté conectado. Esta página puede ser estática o semidinámica.

Pase en el parámetro pagInicio el nombre de la página HTML o de la ruta de acceso HTML a la página.

***Nota:** si la página especificada en el parámetro pagInicio no existe cuando el proceso web accede a ella por primera vez, el servidor web crea y asigna el contenido de la página de inicio predeterminada (ver **Página de inicio por defecto**).*

Para no enviar pagInicio como página de inicio para el proceso web actual, ejecute de nuevo el comando WEB SET HOME PAGE con una cadena vacía ("") pasada en pagInicio.

***Nota:** la página de bienvenida por defecto del servidor web se define en las Propiedades de la base.*

WEB SET HTTP HEADER

WEB SET HTTP HEADER (encab|arrayCamp {; arrayValores})

Parámetro	Tipo		Descripción
encab|arrayCamp	Texto, Array texto	➡	Campo o variable que contiene el encabezado HTTP de la petición o Array de campos del encabezado HTTP
arrayValores	Array texto	➡	Contenido de los campos del encabezado HTTP

Descripción

*El comando **WEB SET HTTP HEADER** permite definir los campos del encabezado HTTP de la respuesta enviada al navegador web por 4D. Sólo tiene efecto en un proceso web.*
Este comando le permite administrar las "cookies".
Hay dos sintaxis disponibles para este comando:

- ***Primera sintaxis: WEB SET HTTP HEADER (encabezado)***
  Pase en el parámetro encab, de tipo variable o campo texto, los campos del encabezado HTTP que quiera definir. Esta sintaxis permite escribir tipos de encabezados tales como "HTTP/1.0 200 OK"+Char(13)+"Set-Cookie: C=HELLO". Los campos de encabezado deben estar separados por un retorno de carro o una secuencia cr/lf (retorno de carro + retorno de línea), bajo Windows y Mac OS, 4D se encarga del formato de la respuesta.

Este es un ejemplo de una "cookie" personalizada:

```
C_TEXT($vTcookie)
$vTcookie:="Set-Cookie: USER="+String(Abs(Random))+"; PATH=/"
WEB SET HTTP HEADER($vTcookie)
```

***Nota:** el comando no aceptará una constante de tipo texto literal en el parámetro encab; debe ser una variable o campo 4D.*
Para mayor información sobre la sintaxis a aplicar, por favor consulte R.F.Cs (Request For Comments) en: http://www.w3c.org.

- ***Segunda sintaxis**: **WEB SET HTTP HEADER** (arrayCamp; arrayValores)*
  El encabezado HTTP está definido con la ayuda de dos arrays de texto, arrayCamp y arrayValores. El encabezado se escribirá así:

```
fieldArray{1}:="X-VERSION"
fieldArray{2}:="X-STATUS"
fieldArray{3}:="Set-Cookie"
fieldArray{4}:="Server"

valueArray{1}:="HTTP/1.0"*
valueArray{2}:="200 OK"*
valueArray{3}:="C=HELLO"
valueArray{4}:="North_Carolina"
```

** Los dos primeros elementos son la primera línea de la respuesta. Cuando se introducen, deben ser los elementos 1 y 2 de los arrays. Sin embargo, es posible omitirlos y escribir únicamente lo siguiente (4D se encargará del formato del encabezado):*

```
fieldArray{1}:="Set-Cookie"
valueArray{1}:="C=HELLO"
```

*Si no especifica un estado, automáticamente será HTTP/1.0 200 OK. El campo **Server** es por defecto "4D/<version>".*
*Los campos **Date** y **Content-Length** siempre son definidos por defecto por 4D.*

WEB SET OPTION

WEB SET OPTION (selector ; valor)

Parámetro	Tipo		Descripción
selector	Entero largo	➡	Código de la opción a modificar
valor	Entero largo, Texto	➡	Valor de la opción

Descripción

*El comando **WEB SET OPTION** modifica el valor actual de las diferentes opciones de funcionamiento del servidor web de 4D.*

Los cambios realizados en estas opciones se conservan si el servidor 4D Web se detiene y reinicia, sin embargo, no se conservan si la aplicación 4D se detiene y se reinicia.

*En el parámetro selector, pase una de las constantes del tema **Servidor web** y pase el nuevo valor de la opción en valor:*

Constante	Tipo	Valor	Comentario
Web character set	*Entero largo*	*17*	***Alcance**: 4D local, 4D Server* ***Descripción**: conjunto de caracteres que el servidor Web 4D (con 4D en modo local y 4D Server) utiliza para comunicarse con los navegadores web que se conectan a base. El valor por defecto depende del lenguaje del sistema operativo. Este parámetro se define en las Propiedades de la base.* ***Valores**: los valores posibles dependen del modo de ejecución de la base relativos al conjunto de caracteres.* • *Modo Unicode: cuando la aplicación se ejecuta en modo Unicode, los valores a pasar para este parámetro son los identificadores de conjunto de caracteres (MIBEnum longint o Name string, identificadores definidos por IANA, consulte: http://www.iana.org/assignments/character-sets). Está es la lista de los identificadores correspondientes a los conjuntos de caracteres que admite el servidor Web de 4D:* *4=ISO-8859-1* *12=ISO-8859-9* *13=ISO-8859-10* *17=Shift-JIS* *2024=Windows-31J* *2026=Big5* *38=euc-kr* *106=UTF-8* *2250=Windows-1250* *2251=Windows-1251* *2253=Windows-1253* *2255=Windows-1255* *2256=Windows-1256* • *Modo compatibilidad ASCII:* *0: Occidental* *1: Japonés* *2: Chino* *3: Coreano* *4: Definido por el usuario* *5: Reservado* *6: Europa central* *7: Cirílico* *8: Árabe* *9: Griego* *10: Hebreo* *11: Turco* *12: Nórdico*
Web Client IP address to listen	*Entero largo*	*23*	***Alcance**: todas las máquinas remotas 4D* ***Valores posibles**: ver Web IP address to listen* ***Descripción**: se utiliza para especificar este parámetro para todas las máquinas 4D remotas utilizadas como servidores web (aplicado al servidor web remoto únicamente).*
Web debug log	*Entero largo*	*84*	***Alcance:** servidor web local* ***Nota:** si se reinicia el servidor HTTP, se utiliza un nuevo archivo de historial* ***Descripción:** le permite obtener o definir el estado del archivo de historial de peticiones HTTP del servidor Web 4D. Cuando se activa, este archivo, llamado "**HTTPDebugLog_nn.txt**", se guarda en la carpeta "Logs" de la aplicación (nn es el número de archivo). Es útil para problemas de depuración relacionados con el servidor web. Registra cada petición y cada respuesta en modo raw. La totalidad de las peticiones, encabezados, se registran; opcionalmente, también se pueden registrar partes del cuerpo. Para mayor información sobre archivos HTTPDebugLog, consulte la sección **Anexo E: Descripción de archivos de historial**.* ***Valores**: una de las constantes con el prefijo "wdl" (consulte las descripciones de estas constantes en este tema).* ***Valor por defecto**: 0 (no activado)*
Web HSTS enabled	*Entero largo*	*86*	***Alcance**: 4D local, 4D Server.* ***Descripción**: estado HTTP Strict Transport Security (HSTS). HSTS permite que el servidor Web 4D declare que los navegadores solo deberían interactuar con él a través de conexiones HTTPS seguras. Una vez activado, el servidor web 4D agregará automáticamente información relacionada con HSTS a todos los encabezados de respuesta.Los navegadores registrarán la información HSTS la primera vez que reciban una respuesta del servidor web 4D, luego toda solicitud HTTP futura se transformará automáticamente en solicitudes HTTPS. El tiempo que el navegador almacena esta información se especifica con el selector Web HSTS max age. HSTS requiere que HTTPS esté habilitado en el servidor. HTTP también debe estar habilitado para permitir las conexiones iniciales del cliente.* ***Valores posibles**: 0 (desactivado, predeterminado) o 1 (activado)* ***Nota:** el servidor web 4D debe reiniciarse para que se aplique esta configuración.*

Constante	Tipo	Valor	Comentario
Web HSTS max age	*Entero largo*	*87*	***Alcance**: 4D local, 4D Server* ***Descripción**: especifica el tiempo máximo (en segundos) que HSTS está activo para cada conexión de cliente nuevo. Esta información se almacena en el lado del cliente durante la duración especificada.* ***Valores posibles**: Entero largo (segundos)* ***Valor por defecto**: 63072000 (2 años)* ***Atención:** una vez que HSTS esté habilitado, las conexiones cliente continuarán usando este mecanismo por la duración especificada. Cuando esté probando sus aplicaciones, se recomienda establecer una duración corta para poder cambiar entre los modos de conexión segura y no segura si es necesario.*
Web HTTP compression level	*Entero largo*	*50*	***Alcance**: Servidor web local* ***Descripción**: nivel de compresión para todos los intercambios HTTP comprimidos efectuados para el servidor HTTP de 4D (peticiones cliente o respuestas servidor, Web y servicio web). Este selector permite optimizar los intercambios con un enfoque en la velocidad de ejecución (menor compresión) o la cantidad de compresión (menor velocidad). La elección de un valor depende del tamaño y la naturaleza de los datos intercambiados. Pase de 1 a 9 en el parámetro valor, 1 es la compresión más rápida y 9 la más alta. También puede pasar -1 para obtener un compromiso entre velocidad y tasa de compresión. El nivel de compresión por defecto es 1 (compresión rápida).* ***Valores posibles**: 1 a 9 (1 = más rápido, más comprimido = 9) o -1 = mejor compromiso.*
Web HTTP compression threshold	*Entero largo*	*51*	***Alcance**: Servidor HTTP local* ***Descripción**: en intercambios HTTP optimizados, límite de tamaño de petición por debajo del cual los intercambios no deben comprimirse. Esta opción es útil para evitar pérdidas de tiempo de máquina para comprimir intercambios muy pequeños.* *Pase en valor un tamaño en bytes. Por defecto, el límite de compresión se establece en 1024 bytes.* ***Valores posibles**: todo valor de tipo entero largo. El parámetro valor contiene una tamaño expresado en bytes. Por defecto, el umbral de compresión está definido en 1024 bytes.*
Web HTTP enabled	*Entero largo*	*88*	***Alcance**: 4D local, 4D Server* ***Descripción**: estados para comunicación sobre HTTP.* ***Valores posibles**: 0 (desactivado) o 1 (activado)*
Web HTTP TRACE	*Entero largo*	*85*	***Alcance**: servidor web local* ***Descripción**: le permite activar o desactivar el método HTTP TRACE en el servidor web de 4D. Por razones de seguridad, a partir de 4D v15 R2, por defecto, el servidor web 4D rechaza peticiones TRACE HTTP con un error 405 (ver desactivación de HTTP TRACE). Si es necesario, puede activar el método HTTP TRACE para la sesión pasando esta constante con el valor 1. Cuando se activa esta opción, el servidor web 4D responde a las solicitudes HTTP TRACE con la línea de petición, el encabezado y el cuerpo.* ***Valores posibles**: 0 (desactivado) o 1 (activado)* ***Valor por defecto**: 0 (desactivado)*
Web HTTPS enabled	*Entero largo*	*89*	***Alcance**: 4D local, 4D Server* ***Descripción**: estado para comunicación sobre HTTPS.* ***Valores posibles**: 0 (desactivado) o 1 (activado)*
Web HTTPS port ID	*Entero largo*	*39*	***Alcance**: 4D local, 4D Server* ***Valores posibles**: 0 a 65535* ***Descripción**: número del puerto TCP utilizado por el servidor web de 4D en modo local y de 4D Server para conexiones seguras vía TLS (protocolo HTTPS). El número de puerto HTTPS se define en la página "Web/Configuración" de la caja de diálogo Propiedades de la base.* *Por defecto, el valor es 443 (valor estándar). Puede utilizar las constantes del tema **Números de puerto TCP** para el parámetro valor.*
Web inactive process timeout	*Entero largo*	*78*	***Alcance**: servidor web local* ***Descripción**: permite modificar el timeout del proceso utilizado para la sesión (opción relativa al proceso). Después del timeout, el proceso se elimina en el servidor, se llama al **Método base On Web Close Process** y luego el contexto de la sesión se destruye.* ***Valores**: Entero largo (minutos)* ***Valores por defecto**: 480 minutos (pase 0 para restablecer el valor por defecto)*
Web inactive session timeout	*Entero largo*	*72*	***Alcance**: servidor web local* ***Descripción**: permite modificar la duración de vida de las sesiones inactivas (duración definida en cookie). Al final de este periodo, la cookie de sesión expira y no se envía más al cliente HTTP.* ***Valores**: Entero largo (minutos)* ***Valores por defecto**: 480 minutos (pase 0 para restablecer el valor por defecto)*
Web IP address to listen	*Entero largo*	*16*	***Alcance**: 4D local, 4D Server* ***Descripción**: dirección IP en la que el servidor web debe recibir las peticiones HTTP con 4D en modo local y 4D Server. Por defecto, ninguna dirección se especifica. Este parámetro se define en las Propiedades de la base. Este selector es útil en el caso de los servidores web 4D compilados y fusionados con 4D Desktop (no hay acceso al modo Diseño).* ***Valores posibles**: dirección IP en forma de cadena. Ambos formatos cadena IPv6 (por ejemplo, "2001:0db8:0000:0000:0000:ff00:0042:8329") y los formatos cadena IPv4 (por ejemplo, "123.45.67.89") son compatibles.* ***Nota:** por compatibilidad, las direcciones IPv4 expresadas en como longitudes hexadecimales (obsoletas) aún son compatibles.*

Constante	Tipo	Valor	Comentario
Web keep session	*Entero largo*	*70*	***Alcance**: servidor web local* ***Descripción**: permite activar o desactivar el modo de gestión de las sesiones (descrito en la sección **Gestión de las sesiones web**).* ***Valores**: 1 (activar modo) ó 0 (desactivar modo)* ***Valor por defecto**: 1 para bases creadas en la versión 13, 0 para bases convertidas. Note que este modo activa igualmente el mecanismo de reutilización de los contextos temporales en modo remoto. Para mayor información sobre este mecanismo, consulte la descripción de esta opción en la sección **Parámetros del servidor web**.*
Web log recording	*Entero largo*	*29*	***Alcance**: 4D local 4D Server* ***Descripción**: inicia o detiene el registro de peticiones solicitudes Web recibida por el servidor web de 4D en modo local o 4D Server. Por defecto, el valor es 0 (no hay registro de peticiones).* *El historial de las peticiones web se guarda en un archivo texto llamado "logweb.txt" que se ubica automáticamente en la carpeta Logs de la base, junto al archivo de estructura. El formato de este fichero es determinado por el valor que se pase. Para más información sobre los diferentes formatos de historial de las peticiones, consulte la sección [#title id= "2833"/]. La activación de este archivo también se puede definir en la página "Web/Avanzado" de las Preferencias de 4D.* ***Valores posibles**: 0 = No guardar (por defecto), 1 = Registrar en formato CLF, 2 = Registrar en formato DLF, 3 = Registrar en formato DLF, 4 = Guardar en formato WLF.* ***Atención**: los formatos 3 y 4 formatos son formatos personalizados, los contenidos deben ser definidos de antemano en las Preferencias de la aplicación, página "Web/Formato del historial". Si usted utiliza uno de estos formatos sin que sus campos hayan sido seleccionados, el archivo de las peticiones no se generará.*
Web max concurrent processes	*Entero largo*	*18*	***Alcance**: 4D local, 4D Server* ***Descripción**: límite estrictamente superior del número de procesos web de todo tipo aceptados por el servidor web con 4D en modo local y 4D Server. Cuando se alcanza el número límite (menos uno), 4D no crea un nuevo proceso y devuelve el mensaje "Servidor no disponible" (estado HTTP 503 - Servicio no disponible) a toda nueva petición.* *Este parámetro previene la saturación del servidor web de 4D que puede ocurrir durante un envío masivo de peticiones o de una demanda excesiva de creación de contextos. También puede definirse en la caja de diálogo de la Propiedades de la base.* *En teoría, el número máximo de procesos web es el resultado de dividir la memoria disponible / tamaño de la pila de un proceso web. Otra solución es ver la información sobre los procesos web que se muestra en el Explorador de ejecución: se indican el número actual de procesos web y el número máximo alcanzado desde el inicio del servidor web.* ***Valores**: todo valor entre 10 y 32 000. El valor por defecto es 100.*
Web max sessions	*Entero largo*	*71*	***Alcance**: servidor web local* ***Descripción**: permite limitar el número de sesiones simultáneas. Cuando se alcanza el número definido, la sesión más antigua se cierra (y se llama al **Método base On Web Close Process**) si el servidor web necesita crear una nueva.* ***Valores posibles:** Entero largo. El número de sesiones simultáneas no puede superar el número total de procesos web (opción Web Max Concurrent Processes, 100 por defecto)* ***Valores por defecto**: 100 (pase 0 para restablecer el valor por defecto)*
Web maximum requests size	*Entero largo*	*27*	***Alcance**: 4D local, 4D Server* ***Descripción**: tamaño máximo (en bytes) de las peticiones HTTP entrantes (POST) que el servidor web está autorizado a tratar. Por defecto, el valor predeterminado es 2 000 000, es decir, un poco menos de 2 MB. El valor máximo (2 147 483 648) significa en la práctica que ningún límite se establece.* *Esta configuración evita la saturación del servidor web, causadas por peticiones entrantes muy grandes. Cuando una petición llega al límite, el servidor web de 4D la rechaza.* ***Valores posibles**: 500 000 a 2 147 483 648.*
Web port ID	*Entero largo*	*15*	***Alcance**: 4D en modo local y 4D Server.* ***Descripción**: establece u obtiene el número del puerto TCP utilizado por el servidor web 4D con 4D en modo local y 4D Server. Por defecto, el valor es 80. El número de puerto TCP se define en la página "Web/Configuración" de la caja de diálogo Propiedades de la base. Puede utilizar una de las constantes del tema **Números de puerto TCP** para el parámetro valor. Este selector es útil en el marco de servidores web 4D que se compilan y fusionan utilizando 4D de escritorio (sin acceso al entorno Diseño).* ***Valores posibles**: para obtener más información sobre el número de puerto TCP, consulte la sección **Parámetros del servidor web**.* ***Valor por defecto**: 80*
Web session cookie domain	*Entero largo*	*81*	***Alcance**: Servidor web local* ***Descripción**: define u obtiene el valor del campo "dominio" de la cookie de sesión. Este selector (así como el selector 82) es útil para controlar el alcance de las cookies de sesión: si configura, por ejemplo, el valor "/*.4d.fr" para este selector, el cliente sólo enviará una cookie cuando la petición se dirige al dominio ".4d.fr", que excluye los servidores que alojan los datos estáticos externos.* ***Valores posibles**: Texto*
Web session cookie name	*Entero largo*	*73*	***Alcance**: servidor web local* ***Descripción**: permite definir el nombre de la cookie utilizada para almacenar el ID de la sesión.* ***Valores**: Texto* ***Valores por defecto**: "4DSID" (pase una cadena vacía para restablecer el valor por defecto)*

Constante	Tipo	Valor	Comentario
Web session cookie path	Entero largo	82	**Alcance**: servidor web local **Descripción**: define u obtiene el valor del campo "path" de la cookie de sesión. Este selector (así como elselector 81) es útil para controlar el alcance de las cookies de sesión: si configura, por ejemplo, el valor "/4DACTION" para este selector, el cliente deberá enviar sólo una cookie para peticiones dinámicas que comiencen con 4DACTION , y no para las imágenes, páginas estáticas, etc. **Valores posibles**: Texto
Web session enable IP address validation	Entero largo	83	**Alcance**: servidorWeb Local **Descripción**: Activa o desactiva la validación de las direcciones IP para las cookies de sesión. Por razones de seguridad, por defecto, el servidor web de 4D verifica la dirección IP de cada solicitud que contiene una cookie de sesión y la rechaza si esta dirección no coincide con la dirección IP utilizada para crear la cookie. En algunas aplicaciones específicas, es posible que desee desactivar esta validación y aceptar las cookies de sesión, incluso cuando sus direcciones IP no coincidan. Por ejemplo, cuando los dispositivos móviles cambian entre redes WiFi y 3G/4G, su dirección IP cambia. En este caso, debe pasar 0 en esta opción para permitir que los clientes puedan seguir utilizando sus sesiones web incluso cuando las direcciones IP cambien. Tenga en cuenta que esta configuración reduce el nivel de seguridad de la aplicación. Cuando se modifica, esta configuración es efectiva inmediatamente (no es necesario reiniciar el servidor HTTP). **Valores posibles**: 0 (desactivado) o 1 (activado) **Valor por defecto**: 1 (las direcciones IP son verificadas)

Al utilizar el selector Web debug log, puede pasar una de las siguientes constantes en el parámetro valor:

Constante	Tipo	Valor	Comentario
wdl disable	Entero largo	0	El archivo de historial de peticiones HTTP Web está desactivado
wdl enable with all body parts	Entero largo	7	El archivo de historial de peticiones HTTP Web está activado con el cuerpo de la respuesta y la respuesta
wdl enable with request body	Entero largo	5	El archivo de historial de peticiones HTTP Web está activado con el cuerpo de la respuesta únicamente
wdl enable with response body	Entero largo	3	El archivo de historial de peticiones HTTP Web está activado con el cuerpo de la respuesta únicamente
wdl enable without body	Entero largo	1	El archivo de historial de peticiones HTTP Web está desactivado sin el cuerpo (el tamaño del cuerpo se entrega en este caso)

Ejemplo

La activación del archivo de historial de depuración de las peticiones HTTP sin las partes del cuerpo:

```
WEB SET OPTION(Web debug log;wdl enable without body)
```

Una entrada registrada se ve así:

```
# REQUEST
# SocketID: 1592
# PeerIP: 127.0.0.1
# PeerPort: 54912
# TimeStamp: 39089388
#ConnectionID: 9808E3B4B06E4EB5A60E9A3FC69116BD
#SequenceNumber:5
GET /4DWEBTEST HTTP/1.1
Accept: text/html,(...)
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: 127.0.0.1
User-Agent: 4D_HTTP_Client/0.0.0.0

# RESPONSE
# SocketID: 1592
# PeerIP: 127.0.0.1
# PeerPort: 54912
# TimeStamp: 39089389 (elapsed time: 1 ms)
#ConnectionID: 9808E3B4B06E4EB5A60E9A3FC69116BD
#SequenceNumber:6
HTTP/1.1 200 OK
Accept-Ranges: bytes
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 3555
Content-Type: text/plain; charset=UTF-8
Date: Thu, 20 Apr 2017 10:51:29 GMT
Expires: Thu, 20 Apr 2017 10:51:29 GMT
```

Server: 4D/16.0.1

[Body Size: 3555]

WEB SET ROOT FOLDER

WEB SET ROOT FOLDER (carpetaRaiz)

Parámetro	Tipo		Descripción
carpetaRaiz	Cadena	➡	Ruta de acceso de la carpeta raíz del servidor Web

Descripción

*El comando **WEB SET ROOT FOLDER** se utiliza para modificar la carpeta raíz por defecto en la cual 4D busca los archivos HTML pedidos al servidor web.*

*Este comando no tiene en cuenta la carpeta raíz HTML por defecto, definida en la Propiedades de la base. Para mayor información sobre esta carpeta, consulte la sección **Seguridad de las conexiones**.*

La ubicación de la carpeta raíz puede expresarse en sintaxis HTML (tipo URL), o en sintaxis sistema (ruta absoluta):

- *Sintaxis HTML: los nombres de carpetas se separan por una barra oblicua ("/"), sin importar la plataforma utilizada.*
- *Sintaxis sistema: ruta de acceso absoluta (“nombre largo”) respetando la sintaxis de la plataforma actual, por ejemplo:*

- (Mac OS) Disco:Aplicaciones:miserv:carpeta
- (Windows) C:\Aplicaciones\miserv\carpeta

Notas:

- *Será necesario reiniciar el servidor web para que la nueva carpeta de raíz sea tenida en cuenta.*
- *Puede saber en cualquier momento la ubicación de la carpeta raíz utilizando el comando Get 4D folder.*

*Si especifica una ruta de acceso invalida, se generará un error del administrador archivos del sistema operativo. Puede interceptar el error con un método **ON ERR CALL**. Si el método muestra una caja de diálogo de alerta o un mensaje, aparecerá del lado del navegador.*

WEB START SERVER

WEB START SERVER
Este comando no requiere parámetros

Descripción

*El comando **WEB START SERVER** inicia el servidor web de la aplicación 4D en la cual se ejecuta (4D o 4D Server). La base es por lo tanto publicada en su red Intranet o en el Internet.*
*Si el servidor web se inicia correctamente, **OK** toma el valor 1, de lo contrario toma el valor 0 (cero). Por ejemplo, si el protocolo de red TCP/IP no está correctamente configurado, **OK** toma el valor 0.*

Variables y conjuntos del sistema

*Si el servidor web se inicia correctamente, **OK** toma el valor 1; de lo contrario **OK** toma el valor 0.*

WEB STOP SERVER

WEB STOP SERVER
Este comando no requiere parámetros

Descripción

*El comando **WEB STOP SERVER** detiene el servidor web de la aplicación 4D en la cual ha sido ejecutado (4D o 4D Server). Si se ha iniciado el servidor web, todas las conexiones web se interrumpen y todos los procesos web terminan.*
Si el servidor web no se ha iniciado, el comando no hace nada.

WEB Validate digest

WEB Validate digest (nomUsuario ; contraseña) -> Resultado

Parámetro	Tipo		Descripción
nomUsuario	Texto	➡	Nombre del usuario
contraseña	Texto	➡	Contraseña del usuario
Resultado	Booleano	⮌	True=Autenticación correcta, False=Falla de la autenticación

Descripción

*El comando **WEB Validate digest** permite verificar la validez de la información de identificación (nombre y contraseña) suministrada por un usuario que se conecta al servidor web. Este comando debe utilizarse en el **Método de base On Web Authentication** en el contexto de una autenticación web en modo Digest (ver la sección).*

Pase en los parámetros nomUsuario y contraseña, la información de identificación del usuario almacenada localmente. El comando utiliza esta información para generar un valor que se compare con la información enviada por el navegador web.

Si los valores son idénticos, el comando devuelve True. De lo contrario, devuelve False.

Puede utilizar este mecanismo para administrar y mantener por programación su propio sistema de acceso seguro al servidor web. Note que la validación Digest no puede utilizarse en conjunto con las contraseñas 4D.

***Nota:** si el navegador no soporta la autenticación Digest, se devuelve un error (error de autenticación).*

Ejemplo

Ejemplo de método de base On Web Authentication en modo Digest:

```
  ` Método de base On Web Authentication
C_TEXT($1;$2;$5;$6;$3;$4)
C_TEXT($usuario)
C_BOOLEAN($0)
$0:=False
$usuario:=$5
  `Por razones de seguridad, rechazar los nombres que contengan @
If(WithWildcard($usuario))
   $0:=False
  `El método WithWildcard se describe en la sección "Método de base On Web Authentication"
Else
   QUERY([WebUsers];[WebUsers]Usuario=$usuario)
   If(OK=1)
      $0:=Validate Digest Web Password($usuario;[WebUsers]contraseña)
   Else
      $0:=False  `Usuario inexistente
   End if
End if
```

_o_SET CGI EXECUTABLE

_o_SET CGI EXECUTABLE (url1 {; url2})

Parámetro	Tipo		Descripción
url1	Cadena	➡	***inutilizado
url2	Cadena	➡	***inutilizado

Comando desactivado

Este comando está desactivado a partir de 4D v13. Si se llama, se genera el error 33 ("Método o función no implementado").

_o_SET WEB DISPLAY LIMITS

_o_SET WEB DISPLAY LIMITS (numeroRegistros {; numeroPaginas {; refImag}})

Parámetro	Tipo		Descripción
numeroRegistros	*Entero largo*	➡	**** Desactivado ****
numeroPaginas	*Entero largo*	➡	**** Desactivado ****
refImag	*Entero largo*	➡	**** Desactivado ****

Comando desactivado

Comando desactivado a partir de 4D v13. Si se llama, se genera el error 33 ("Método o función no implementado").

_o_SET WEB TIMEOUT

_o_SET WEB TIMEOUT (timeout)

Parámetro	Tipo		Descripción
timeout	Entero largo	➡	Tiempo de desconexión de la conexión web expresado en segundos

Comando desactivado

Este comando se desactiva a partir de 4D v13. Si se llama, se genera el error 33 ("Método o función no implementado").

_o_Web Context

_o_Web Context -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano	➲	Siempre devuelve False

Comando desactivado

Este comando está desactivado a partir de 4D v13. Si se llama, se genera el error 33 ("Método o función no implementado").

SQL

- Presentación de los comandos del tema SQL
- Método de base On SQL Authentication
- Begin SQL
- End SQL
- Get current data source
- GET DATA SOURCE LIST
- Is field value Null
- QUERY BY SQL
- SET FIELD VALUE NULL
- SQL CANCEL LOAD
- SQL End selection
- SQL EXECUTE
- SQL EXECUTE SCRIPT
- SQL EXPORT DATABASE
- SQL EXPORT SELECTION
- SQL GET LAST ERROR
- SQL GET OPTION
- SQL LOAD RECORD
- SQL LOGIN
- SQL LOGOUT
- SQL SET OPTION
- SQL SET PARAMETER
- START SQL SERVER
- STOP SQL SERVER
- _o_USE EXTERNAL DATABASE
- _o_USE INTERNAL DATABASE

Presentación de los comandos del tema SQL

4D incluye un motor SQL integrado. El programa también incluye un servidor SQL que puede ser consultado por otras aplicaciones 4D o de terceras partes (vía el piloto OBDC de 4D).
La documentación SQL de 4D está compuesta de dos partes:

- *La **Guía de referencia SQL de 4D** (**Manual de SQL**). Este manual describe los diferentes modos de acceso al motor SQL de 4D, la configuración del servidor SQL como también los comandos y palabras claves utilizables en las peticiones SQL (por ejemplo **SELECT** o **UPDATE**). Consulte este manual para mayor información sobre la utilización del lenguaje SQL en 4D.*
- *El **tema SQL del manual "Lenguaje" de 4D** (**SQL**). Este tema agrupa los diferentes comandos internos de 4D relativos al uso de SQL en 4D:*
  - *Control del servidor SQL: **START SQL SERVER** y **STOP SQL SERVER***
  - *Acceso directo al motor SQL integrado: **SET FIELD VALUE NULL**, **Is field value Null**, **QUERY BY SQL***
  - *Gestión de las conexiones a fuentes de datos externas o internas (SQL pass-through): **GET DATA SOURCE LIST**, **Get current data source**, **SQL LOGIN**, **SQL LOGOUT**.*
  - *Comandos de alto nivel para la manipulación de datos en el marco de conexiones SQL directas o vía ODBC: **Begin SQL**, **End SQL**, **SQL CANCEL LOAD**, **SQL LOAD RECORD**, **SQL EXECUTE**, **SQL End selection**, **SQL SET OPTION**, **SQL SET PARAMETER**, **SQL GET LAST ERROR**, **SQL GET OPTION**.*

Cómo funcionan los comandos de alto nivel SQL

Los comandos SQL integrados de 4D comienzan con el prefijo "SQL" e implementan los siguientes principios:

- *A menos de que se indique lo contrario, puede utilizar estos comandos con el motor SQL interno 4D o en una conexión externa que se abre directamente o vía ODBC. El comando **SQL LOGIN** permite definir el tipo de conexión a abrir.*
- *El alcance de una conexión es el proceso. Si quiere administrar simultáneamente varias conexiones, debe iniciar un proceso por **SQL LOGIN**.*
  *El comando **SQL CANCEL LOAD** permite ejecutar varias solicitudes SELECT en la misma conexión.*
- *Puede interceptar los errores ODBC generados durante la ejecución de uno de los comandos SQL de alto nivel utilizando el comando **ON ERR CALL**. El comando **SQL GET LAST ERROR** puede utilizarse en este caso para obtener información adicional.*

Soporte del estándar of ODBC

El estándar ODBC (Open DataBase Connectivity) define una librería de funciones estandarizadas. Estas funciones permiten a una aplicación como 4D acceder a través del lenguaje SQL a todos los sistemas de gestión de datos compatibles con ODBC (bases de datos, hojas de cálculo, otras aplicaciones 4D, etc.).

***Nota**: 4D también permite importar y exportar datos en una fuente ODBC vía los comandos **IMPORT ODBC** and **EXPORT ODBC** o manualmente en modo Diseño. Para mayor información, consulte el Manual de Diseño 4D.*

***Nota**: los comandos SQL de alto nivel de 4D permiten implementar soluciones simples para la comunicación entre las aplicaciones 4D y las fuentes de datos ODBC. Si sus aplicaciones necesitan un soporte más extenso del estándar ODBC, necesitará el plug-in ODBC "bajo nivel" de 4D, **4D ODBC Pro**.*

Correspondencia de los tipos de datos

La siguiente tabla lista las correspondencias establecidas automáticamente por 4D entre los tipos de datos 4D y SQL:

4D Tipo	SQL Tipo
C_STRING	SQL_C_CHAR
C_TEXT	SQL_C_CHAR
C_REAL	SQL_C_DOUBLE
C_DATE	SQL_C_TYPE_DATE
C_TIME	SQL_C_TYPE_TIME
C_BOOLEAN	SQL_C_BIT
C_INTEGER	SQL_C_SHORT
C_LONGINT	SQL_C_SLONG
C_BLOB	SQL_C_BINARY
C_PICTURE	SQL_C_BINARY
C_GRAPH	SQL_C_BINARY

***Nota:** los datos de tipo objeto (**C_OBJECT**) no son soportados por el motor SQL de 4D.*

Referenciar las expresiones 4D en las solicitudes SQL

*4D ofrece dos maneras de insertar expresiones 4D (variables, arrays, campos, expresiones válidas) en las solicitudes SQL: la asociación directa y la definición de parámetros utilizando **SQL SET PARAMETER**.*

La asociación directa se puede efectuar de dos formas:

- *Insertando el nombre del objeto 4D a utilizar entre los caracteres << y >> en el texto de la solicitud.*
- *Precediendo la referencia con dos puntos":".*
  Ejemplos:

```
SQL EXECUTE("INSERT INTO emp (empnum,enombre) VALUES (<<vEmpnum>>,<<vEnombre>>)")
SQL EXECUTE("SELECT edad FROM Persona WHERE nombre= :vNombre")
```

Nota: *en modo compilado, no puede utilizar referencias a variables locales (que comienzan por $).*

En estos ejemplos, los valores actuales de las variables 4D vEmpnum, vEnombre y Vnombre reemplazarán los parámetros cuando la petición se ejecute. Esta solución también funciona con campos y arrays 4D.

Esta sintaxis de fácil utilización, presenta el inconveniente de no cumplir el estándar SQL y de no permitir la utilización de parámetros de salida. Para remediar esto, puede utilizar el comando ***SQL SET PARAMETER****. Este comando permite definir cada objeto 4D a integrar en una solicitud así como también su modo de utilización (entrada, salida o ambos). Entonces la sintaxis producida es estándar. Para mayor información, consulte la descripción del comando* ***SQL SET PARAMETER.***

1. Este ejemplo ejecuta una solicitud SQL que utiliza directamente los arrays 4D asociados:

```
ARRAY TEXT(MiArrayText;10)
ARRAY LONGINT(MiArrayEnteroLargo;10)

For(vContador;1;Size of array(MiArrayText))
   MiArrayText{vContador}:="Text"+String(vContador)
   MiArrayEnteroLargo{vContador}:=vContador
End for
SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (campo_alfa, campo_enterolargo) VALUES (<<MiArrayText>>, <<MiArrayEnteroLargo>>)"
SQL EXECUTE(SQLStmt)
```

2. Este ejemplo permite ejecutar una petición SQL utilizando directamente los campos 4D asociados:

```
ALL RECORDS([Tabla 2])
SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (campo_alfa, campo_enterolargo) VALUES (<<[Tabla 2]Campo1>"+">,<<[Tabla2]Campo2>>)"
SQL EXECUTE(SQLStmt)
```

3. Este ejemplo permite ejecutar una búsqueda SQL pasando directamente una variable utilizando un puntero derreferenciado:

```
C_LONGINT($vLong)
C_POINTER($vPuntero)
$vLong:=1
$vPuntero:=->$vLong
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT Col1 FROM TEST WHERE Col1=:$vPuntero"
SQL EXECUTE(SQLStmt)
```

Uso de variables en modo compilado

En modo compilado, puede utilizar referencias de variables locales (comenzando por el carácter $) en instrucciones SQL bajo ciertas condiciones:

- *Puede utilizar variables locales dentro de una secuencia* ***Begin SQL*** */* ***End SQL****, excepto con el comando* ***EXECUTE IMMEDIATE****;*
- *Puede utilizar variables locales con el comando* ***SQL EXECUTE*** *cuando estas variables se utilizan directamente en el parámetro de petición SQL y no vía las referencias.*
  Por ejemplo, el siguiente código funciona en modo compilado:

```
SQL EXECUTE("select * from t1 into :$myvar") // funciona en modo compilado
```

El siguiente código generará un error en modo compilado:

```
C_TEXT(tRequest)
tRequest:="select * from t1 into :$myvar"
SQL EXECUTE(tRequest) // error en modo compilado
```

Recuperación de valores en 4D

La recuperación de los valores en el lenguaje 4D que resulta de las de las consultas SQL se lleva a cabo de dos formas:

- *Utilizando los parámetros adicionales del comando* ***SQL EXECUTE*** *(solución recomendada).*
- *Utilizando la cláusula INTO en la búsqueda SQL misma (solución reservada para casos especiales).*

Visualización del resultado de una petición SQL en un list box

*Es posible poner directamente el resultado de una petición SQL en un list box de tipo array. Esta función ofrece un medio rápido de visualizar el resultado de peticiones SQL. Sólo pueden utilizarse las peticiones de tipo **SELECT**. Este mecanismo no puede utilizarse con una base SQL externa.*

Esto funciona de acuerdo a los siguientes principios:

- *Cree el list box que recibirá los resultados de la petición. La fuente de datos del list box debe ser **Arrays**.*
- *Ejecute la petición SQL de tipo **SELECT** y asigne el resultado a la variable asociada al list box. Puede utilizar las palabras claves **Begin SQL**/**End SQL** (ver el manual de Lenguaje de 4D).*
- *Las columnas del list box son ordenables y modificables por el usuario.*
- *Cada nueva ejecución de una petición **SELECT** con la list box provoca la reinicialización de las columnas (no es posible llenar el mismo list box progresivamente utilizando varias peticiones **SELECT**).*
- *Se recomienda dar al list box el mismo número de columnas que las que tendrá en el resultado de petición SQL. Si el número de columnas del list box es inferior al del necesario para la petición **SELECT**, las columnas se añaden automáticamente. Si el número de columnas del list box es superior al necesario para la petición **SELECT**, se ocultan las columnas innecesarias.*
  ***Nota:** las columnas añadidas automáticamente están relacionadas con las **Variables dinámicas** de tipo array. Estos arrays dinámicos permanecen siempre y cuando el formulario exista. Una variable dinámica se crea igualmente para cada encabezado. Cuando se llama el comando **LISTBOX GET ARRAYS**, el parámetro arrVarCols contiene los punteros a los arrays dinámicos y el parámetro arrVarEncabezados contiene los punteros a las variables de encabezados dinámicos. Si una columna añadida es por ejemplo la quinta columna, su nombre es sql_column5 y su nombre de encabezado es sql_header5.*
- *En modo interpretado, los arrays existentes utilizados por el list box pueden redigitarse automáticamente de acuerdo a los datos enviados por la petición SQL*

Ejemplo
Queremos recuperar todos los campos de la tabla PERSONAS y ubicar su contenido en el list box cuyo nombre de variable es vlistbox. En el método de objeto de un botón (por ejemplo), es suficiente escribir:

```
Begin SQL
   SELECT * FROM PEOPLE INTO <<vlistbox>>
End SQL
```

Método de base On SQL Authentication

---

*El **Método de base On SQL Authentication** puede utilizarse para filtrar las peticiones enviadas al servidor SQL integrado de 4D. Este filtro puede estar basado en el nombre y contraseña como también de manera opcional en la dirección IP del usuario. El desarrollador puede utilizar su propia tabla de usuarios o la de los usuarios 4D para evaluar los identificadores de conexión. Una vez validada la conexión, el comando **CHANGE CURRENT USER** puede utilizarse para controlar el acceso de las peticiones dentro de la base 4D.*

*Cuando existe, el **Método de base On SQL Authentication** es llamado automáticamente por 4D o 4D Server en cada conexión externa al servidor SQL. Por lo tanto el sistema interno de gestión de los usuarios de 4D no está activado. La conexión se acepta sólo si el método de base devuelve **True** en $0 y si el comando **CHANGE CURRENT USER** se ha ejecutado con éxito. Si una de estas condiciones no se cumple, la petición se rechaza.*

***Nota**: la instrucción **SQL LOGIN(SQL_INTERNAL**;$usuario;$contraseña) no llama al **Método de base On SQL Authentication** ya que es una conexión interna en este caso.*

El método de base recibe hasta tres parámetros de tipo Texto, pasados por 4D ($1, $2 y $3) y devuelve un booleano, $0. Esta es la descripción de estos parámetros:

Parámetros	***Tipo***	***Descripción***
$1	*Texto*	*Nombre de usuario*
$2	*Texto*	*Contraseña*
$3	*Texto*	*(opcional) Dirección IP del cliente al origen de la petición (*)*
$0	*Booleano*	*True = petición aceptada, False = petición rechazada*

*(*) 4D devuelve las direcciones IPv4 en un formato híbrido IPv6/IPv4 escrito con un prefijo de 96 bits, por ejemplo ::ffff:192.168.2.34 para la dirección IPv4 192.168.2.34. Para mayor información, consulte la sección **Soporte de IP v6**.*

Debe declarar estos parámetros de esta forma:

```
  ` Método de base On Web Authentication

C_TEXT($1;$2;$3;$4)
C_BOOLEAN($0)
  ` Código para el método
```

La contraseña ($2) se recibe como texto estándar.

*Debe controlar los identificadores de la conexión SQL en el **Método de base On SQL Authentication** . Por ejemplo, puede verificar el nombre y la contraseña utilizando una tabla de usuarios personalizada. Si los identificadores son válidos, pase **True** en $0 para aceptar la conexión y la petición. 4D abre una sesión SQL para el usuario. De lo contrario, pase **False** en $0; en este caso, la conexión se rechaza.*

***Nota:** si el **Método de base On SQL Authentication** no existe, la conexión se evalúa utilizando el sistema integrado de gestión de usuarios de 4D (si está activo, en otras palabras, si una contraseña ha sido asignada al Diseñador). Si este sistema no está activado, los usuarios están conectados con los derechos de acceso del Diseñador (acceso libre).*

*Si pasa **True** en $0, debe llamar exitosamente al comando **CHANGE CURRENT USER** en el **Método de base On SQL Authentication** para que la petición sea aceptada y para que 4D abra una sesión SQL para el usuario.*

El uso de este comando se recomienda porque permite un mayor nivel de seguridad. Esta autenticación virtual tiene la doble ventaja de permitir el control de las acciones de conexión y de ocultar para el exterior los identificadores de la conexión en la sesión SQL 4D.

*Cuando el sistema de contraseñas integrado de 4D no está activo, la ejecución del comando **CHANGE CURRENT USER** no tiene efecto; los usuarios se conectan con los derechos de acceso del Diseñador.*

*Este ejemplo del **Método de base On SQL Authentication** verifica que la petición de conexión provenga de la red interna, valida los identificadores y luego asigna los derechos de accesos "sql_user" para la sesión SQL.*

```
C_TEXT($1;$2;$3;$4)
C_BOOLEAN($0)
  `$1: usuario
  `$2: contraseña
  `{$3: dirección IP del cliente}
ON ERR CALL("SQL_error")
If(DirIPInterna($3))
  `El método DirIPInterna verifica si la dirección IP es interna
   If($1="victor") & ($2="hugo")
      CHANGE CURRENT USER("sql_user";"")
      If(OK=1)
         $0:=True
      Else
         $0:=False
      End if
   Else
      $0:=False
   End if
Else
```

```
    $0:=False
End if
```

Begin SQL

Begin SQL
Este comando no requiere parámetros

Descripción

Begin SQL *es una palabra clave que permite indicar en el editor de métodos el inicio de una secuencia de comandos SQL que debe ser interpretada por la fuente de datos actual del proceso (el motor SQL integrado de 4D o toda fuente especificada vía el comando* ***SQL LOGIN****).*

Una secuencia de comandos SQL comienza por ***Begin SQL*** *y debe terminar con la palabra clave* ***End SQL****.*

Estas palabras claves funcionan de esta forma:

- *Puede poner uno o más bloques de etiquetas* ***Begin SQL****/****End SQL*** *en el mismo método. Puede generar métodos totalmente compuestos por código SQL o mezclar código 4D y código SQL en el mismo método.*
- *Puede escribir varias instrucciones SQL en la misma línea o en diferentes líneas separándolas con punto y coma ";". Por ejemplo, puede escribir::*

```
Begin SQL
   INSERT INTO SALESREPS (NOMBRE, EDAD) VALORES (Enrique,40);
   INSERT INTO SALESREPS (NOMBRE, EDAD) VALORES (Bernardo,35)
End SQL
```

o:

```
Begin SQL
   INSERT INTO SALESREPS (NOMBRE, EDAD) VALORES (Enrique,40);INSERT INTO SALESREPS (NAME, AGE) VALUES (Bernardo,35)
End SQL
```

Note que el ***Depurador*** *4D evaluará el código SQL línea por línea. En algunos casos, puede ser mejor utilizar más de una línea.*

End SQL

End SQL

Este comando no requiere parámetros

Descripción

End SQL *es una palabra clave que indica el fin de una secuencia de comandos SQL en el editor de métodos que debe ser interpretada por el motor SQL integrado de 4D.*

Una secuencia de instrucciones SQL debe estar rodeada por las palabras claves ***Begin SQL*** *y* ***End SQL****. Para mayor información, por favor consulte la descripción de la palabra clave* ***Begin SQL****.*

Get current data source

Get current data source -> Resultado

Parámetro	Tipo		Descripción
Resultado	Cadena		Nombre de la fuente de datos que está siendo utilizada

Descripción

*El comando **Get current data source** devuelve el nombre de la fuente de datos actual de la aplicación. La fuente de datos actual recibe las búsquedas SQL ejecutadas dentro de las estructuras **Begin SQL**/**End SQL**.*

*Cuando la fuente de datos actual es la base 4D local, el comando devuelve la cadena ";DB4D_SQL_LOCAL;", que corresponde al valor de la constante **SQL_INTERNAL** (tema "**SQL**").*

Este comando permite verificar la fuente de datos actual, generalmente antes de ejecutar una búsqueda SQL.

GET DATA SOURCE LIST

GET DATA SOURCE LIST (tipoFuente ; arrayNomsFuentes ; arraydrivers)

Parámetro	Tipo		Descripción
tipoFuente	Entero largo	➡	Tipo de fuente: usuario o sistema
arrayNomsFuentes	Array texto	⬅	Array de nombres de fuentes de datos
arraydrivers	Array texto	⬅	Array de drivers de las fuentes

Descripción

*El comando **GET DATA SOURCE LIST** devuelve en los arrays arrayNomsFuentes y arrayDrivers, los nombres y drivers de las fuentes de datos de tipo tipoFuente definidas en el administrador ODBC del sistema operativo.*

*4D le permite conectarse directamente vía el lenguaje a una fuente de datos ODBC externa y ejecutar búsquedas SQL dentro de una estructura **Begin SQL/End SQL**. Este principio funciona de esta forma: el comando **GET DATA SOURCE LIST** permite obtener la lista de fuentes de datos presentes en el equipo. El comando **SQL LOGIN** permite designar la fuente a utilizar. Luego puede ejecutar las búsquedas SQL utilizando una estructura **Begin SQL/End SQL** en la fuente "actual". Para llevar a cabo nuevas búsquedas utilizando nuevamente el motor interno de 4D, simplemente pase el comando **USE INTERNAL DATABASE**. Para mayor información sobre los comandos SQL en el editor de métodos, consulte el manual 4D SQL.*

*En tipoFuente, pase el tipo de fuente de datos que quiere obtener. Puede utilizar una de las siguientes constantes del tema "**SQL**":*

Constante	Tipo	Valor
User data source	Entero largo	1
System data source	Entero largo	2

***Nota:** este comando tiene en cuenta las fuentes de datos de tipo archivo.*

El comando llena y dimensiona los arrays arrayNomsFuentes y arrayDrivers con los valores correspondientes.

***Nota:** si quiere conectarse a una fuente de datos 4D externa vía ODBC, necesitará tener instalado 4D ODBC Driver en su equipo. Para mayor información, consulte el manual de instalación del driver 4D ODBC.*

Ejemplo

Este ejemplo utiliza una fuente de datos usuario:

```
ARRAY TEXT(arrDSN;0)
ARRAY TEXT(arrDSNDrivers;0)
GET DATA SOURCE LIST(User data source;arrDSN;arrDSNDrivers)
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0 y se genera un error.

Is field value Null

Is field value Null (unCampo) -> Resultado

Parámetro	*Tipo*		*Descripción*
unCampo	*Campo*	→	*Campo a evaluar*
Resultado	*Booleano*	←	*True = campo es NULL, False = campo no es NULL*

Descripción

*El comando **Is field value Null**devuelve **True** si el campo designado por el parámetro unCampo contiene el valor NULL y de lo contrario **False**.*

*El valor NULL es utilizado por el motor SQL de 4D. Para mayor información, consulte el manual **Manual de SQL**.*

*El valor devuelto por este comando sólo tiene sentido si la opción "**Mapear valores NULOS a valores vacíos**" no está seleccionada en la definición del campo del editor de estructura. De lo contrario, siempre devuelve **False**.*

QUERY BY SQL

QUERY BY SQL ({tabla ;} formulaSQL)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla en la cual devolver una selección de registros o Tabla por defecto si este parámetro se omite
formulaSQL	Cadena	➡	Fórmula de búsqueda SQL válida representando la cláusula WHERE de la búsqueda SELECT

Descripción

*El comando **QUERY BY SQL** permite aprovechar directamente el motor SQL integrado de 4D. Este comando puede ejecutar una petición SELECT simple que puede escribirse de esta forma:*

```
SELECT *
   FROM tabla
   WHERE <sqlFormula>
```

unaTabla es el nombre de la tabla pasada en el primer parámetro y formulaSQL la cadena de búsqueda pasada en el segundo parámetro.

Por ejemplo, la siguiente instrucción:

```
([Empleados];"nombre='juan'")
```

es equivalente a la búsqueda SQL:

```
SELECT * FROM Empleados WHERE "nombre='juan'"
```

*El comando **QUERY BY SQL** es similar al comando **QUERY BY FORMULA**. El comando busca registros en la tabla especificada. Modifica la selección actual de unaTabla para el proceso actual y vuelve el primer registro de la nueva selección el registro actual.*

***Nota:** el comando **QUERY BY SQL** no puede utilizarse en el contexto de una conexión SQL externa; él contacta el motor SQL integrado de 4D directamente.*

***QUERY BY SQL** aplica formulaSQL a cada registro de la selección de la tabla. formulaSQL es una expresión booleana que debe devolver **True** o **False**. Como sabe, en el estándar SQL, una condición de búsqueda puede tener un resultado **True**, **False** o **NULL**. Todos los registros (filas) donde la condición de búsqueda devuelva **True** se incluyen en la nueva selección actual.*

*La expresión formulaSQL puede ser simple, como la comparación de un campo (columna) con un valor; o compleja, como un cálculo. Como **QUERY BY FORMULA**, **QUERY BY SQL** puede evaluar la información en las tablas relacionadas (ver el ejemplo 4). formulaSQL debe ser una instrucción SQL válida, conforme con el estándar SQL-2 y con respecto a las limitaciones actuales de implementación del SQL en 4D. Para mayor información sobre soporte SQL en 4D, consulte el manual 4D SQL.*

*El parámetro formulaSQL puede utilizar referencias a expresiones 4D. La sintaxis a utilizar es la misma que para los comandos SQL integrados o el código incluido entre las etiquetas **Begin SQL**/**End SQL**, es decir: <<MiVar>> o :MiVar.*

Para mayor información, consulte la sección .

***Nota:** este comando es compatible con los comandos **SET QUERY LIMIT** y **SET QUERY DESTINATION**.*

Recordatorio**: las referencias a las variables locales no son posibles en modo compilado. Para mayor información sobre la programación SQL en 4D, consulte la sección **Presentación de los comandos del tema SQL

Acerca de relaciones

***QUERY BY SQL** no utiliza relaciones entre tablas definidas en el editor de estructura 4D. Si quiere utilizar los datos relacionados, tendrá que añadir **JOIN** a la búsqueda. Por ejemplo, asumiendo que tenemos la siguiente estructura, en la cual una relación Muchos a Uno está relacionado los campos [Personas]Ciudad con [Ciudades]Nombre:*

```
[Personas]
   Nombre
   Ciudad
[Ciudades]
   Nombre
   Poblacion
```

*Utilizando el comando **QUERY BY FORMULA**, puede escribir:*

```
QUERY BY FORMULA([Personas];[Ciudades]Poblacion>1000)
```

*Utilizando **QUERY BY SQL**, debe escribir la siguiente instrucción, sin importar si la relación existe o no:*

```
QUERY BY SQL([Personas];"personas.ciudad=ciudades.nombre AND ciudades.población>1000")
```

***Nota: QUERY BY SQL** trata las relaciones Uno a Muchos y Muchos a Uno de una manera diferente a la de **QUERY BY FORMULA**.*

Ejemplo 1

Este ejemplo muestra las oficinas con ventas superiores a 100. La búsqueda SQL es:

```
SELECT *
    FROM Oficinas
    WHERE Ventas > 100
```

*Utilizando el comando **QUERY BY SQL**:*

```
C_STRING(30;$formulabusqueda)
$formulabusqueda:="Ventas> 100"
QUERY BY SQL([Oficinas];$formulabusqueda)
```

Ejemplo 2

Este ejemplo muestra las órdenes comprendidas entre 3 000 y 4 000. La búsqueda SQL es:

```
SELECT *
    FROM Ordenes
    WHERE Cantidad BETWEEN 3000 AND 4000
```

*Utilizando el comando **QUERY BY SQL**:*

```
C_STRING(40;$formulabusqueda)
$formulabusqueda:="Cantidad BETWEEN 3000 AND 4000"
QUERY BY SQL([Ordenes];$formulabusqueda)
```

Ejemplo 3

Este ejemplo muestra cómo obtener el resultado de la búsqueda ordenado con un criterio específico. La búsqueda SQL es:

```
SELECT *
    FROM Personas
    WHERE City = 'Paris'
        ORDER BY Nombre
```

*Utilizando el comando **QUERY BY SQL**:*

```
C_STRING(40;$formulabusqueda)
$formulabusqueda:="City= 'Paris' ORDER BY Nombre"
QUERY BY SQL([Personas];$formulabusqueda)
```

Ejemplo 4

Este ejemplo muestra una búsqueda utilizando tablas relacionadas en 4D. En SQL debe utilizar un JOIN para simular esta relación. Consideremos las dos tablas siguientes:

```
[Facturas] con los campos (columnas) siguientes:
    ID_Fact: Entero largo
    Fecha_Fact: Fecha
    Total: Real
[Lineas_Facturas] con las siguientes columnas (campos):
    ID_Linea: Entero largo
    ID_Fact: Entero largo
    Codigo: Alfa (10)
```

Existe una relación Muchos a Uno de [Lineas_Facturas]ID_Fact con [Facturas]ID_Fact.
*Utilizando el comando **QUERY BY FORMULA**, puede escribir:*

```
QUERY BY FORMULA([Lineas_Facturas];([Lineas_Facturas]Codigo="FX-200") & (Month of([Facturas]Fecha_Fact)=4))
```

La búsqueda SQL es:

```
SELECT ID_Linea
    FROM Lineas_Facturas, Facturas
    WHERE Lineas_Facturas.ID_Fact=Facturas.ID_Fact
        AND Lineas_Facturas.Codigo='FX-200'
        AND MONTH(Facturas.Fecha_Fact) = 4
```

*Cuando utiliza el comando **QUERY BY SQL**:*

```
C_STRING(40;$queryFormula)
$formulaBusqueda:="Lineas_Facturas.ID_Fact=Facturas.ID_FactAND Lineas_Facturas.Codigo='FX-200' AND
MONTH(Facturas.Fecha_Fact)=4"
QUERY BY SQL([Lineas_Facturas];$formulaBusqueda)
```

Variables y conjuntos del sistema

Si el formato de la condición de búsqueda es correcto, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0, el resultado del comando es una selección vacía y se devuelve un error. Este error puede ser interceptado por un método instalado utilizando el comando ***ON ERR CALL****.*

SET FIELD VALUE NULL

SET FIELD VALUE NULL (unCampo)

Parámetro	***Tipo***		***Descripción***
unCampo	*Campo*	➡	*Campo al cual atribuir el valor NULL*

Descripción

*El comando **SET FIELD VALUE NULL** asigna el valor NULL al campo designado por el parámetro unCampo.*
El valor NULL es utilizado por el motor SQL de 4D. Para mayor información, consulte el manual de 4D SQL.
Notas:

- *Es posible no permitir el valor Null para los campos 4D al nivel del editor de Estructura (ver el manual de Diseño).*
- ***SET FIELD VALUE NULL** borra el contenido de los campos objeto.*

SQL CANCEL LOAD

SQL CANCEL LOAD
Este comando no requiere parámetros

Descripción

*El comando **SQL CANCEL LOAD** finaliza la solicitud **SELECT** actual e inicializa los parámetros.*
*Este comando se utiliza para ejecutar varias peticiones **SELECT** dentro de la misma conexión (es decir el mismo cursor) iniciada por el comando **SQL LOGIN**.*

Ejemplo

En este ejemplo, se ejecutan dos peticiones en la misma conexión:

```
C_BLOB(Miblob)
C_TEXT(MiTexto)
SQL LOGIN("mysql";"root";"")

SQLStmt:="SELECT blob_field FROM app_testTable"
SQL EXECUTE(SQLStmt;Myblob)
While(Not(SQL End selection))
   SQL LOAD RECORD
End while

 `Reinicialización del cursor
SQL CANCEL LOAD

SQLStmt:="SELECT Nombre FROM Empleado"
SQL EXECUTE(SQLStmt;MyText)
While(Not(SQL End selection))
   SQL LOAD RECORD
End while
```

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente, la variable sistema OK devuelve 1. De lo contrario, devuelve 0.

SQL End selection

SQL End selection -> Resultado

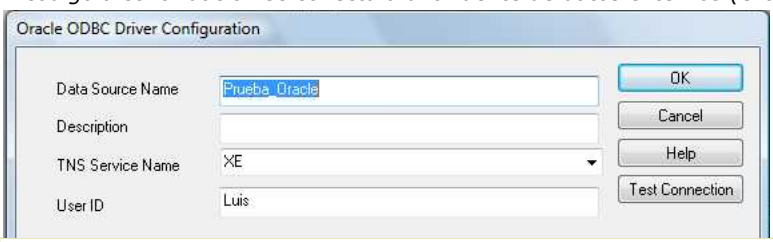
Parámetro	Tipo		Descripción
Resultado	Booleano	➲	El resultado establece los límites alcanzados

Descripción

El comando **SQL End selection** se utiliza para determinar si los límites del resultado obtenido se han alcanzado.

Ejemplo

El código a continuación se conecta a una fuente de datos externos (Oracle) utilizando los siguientes parámetros:




```
C_TEXT(vNombre)

SQL LOGIN("Prueba_Oracle";"Luis";"pwd")
If(OK=1)
    SQL EXECUTE("SELECT ename FROM emp";vNombre)
    While(Not(SQL End selection))
        SQL LOAD RECORD
    End while
    SQL LOGOUT
End if
```

Este código devolverá en la variable 4D vNombre los nombres (ename) almacenados en la tabla emp.

SQL EXECUTE

SQL EXECUTE (instruccionSQL {; objAsoc}{; objAsoc2 ; ... ; objAsocN})

Parámetro	Tipo		Descripción
instruccionSQL	Texto	➡	Comando SQL a ejecutar
objAsoc	Variable, Campo	⬅	Recibe el resultado (si es necesario)

Descripción

*El comando **SQL EXECUTE** se utiliza para ejecutar un comando SQL y asociar el resultado a objetos 4D (arrays, variables o campos).*

Para que el comando pueda ejecutarse, se requiere una conexión válida en el proceso actual.

El parámetro instruccionSQL contiene el comando SQL a ejecutar. objAsoc recibe los resultados. Las variables están asociadas en el orden de secuencia de la columna, lo que significa que las columnas restantes se ignoren.

Si los campos 4D se pasan como parámetros en objAsoc, el comando creará registros y los guardará automáticamente. Los campos 4D deben venir de la misma tabla (no es posible pasar un campo de la tabla 1 y un campo de la tabla 2 en la misma llamada). Si se pasan campos de diferentes tablas, se genera un error.
***Atención:** cuando pase los campos 4D en los parámetros objAsoc y ejecute el comando **SELECT**, siempre son los datos de la fuente 4D remota los que se modifican. Si quiere recuperar datos de una fuente remota localmente, debe utilizar arrays locales intermediarios y llamar al comando **INSERT** (ver el ejemplo 6).*

Si pasa arrays 4D en el parámetro objAsoc, se recomienda declararlos antes de llamar el comando para verificar el tipo de datos procesados. Los arrays se redimensionan automáticamente cuando es necesario.

En el caso de una variable 4D, se recupera un sólo registro a la vez. Los otros resultados se ignoran.

***Nota:** para mayor información sobre el referenciamiento de expresiones 4D en búsquedas SQL, consulte la sección **Presentación de los comandos del tema SQL**.*

Ejemplo 1

En este ejemplo, obtendremos la columna ename de la tabla emp de la fuente de datos externos. El resultado se almacena en el campo 4D [Empleados]Nombre. Los registros 4D se crean automáticamente:

```
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTE(SQLStmt;[Empleados]Nombre)
SQL LOAD RECORD(SQL all records)
```

Ejemplo 2

Para controlar la creación de registros, es posible incluir el código en una transacción y validarla únicamente si la operación prueba ser satisfactoria:

```
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT campo _alfa FROM ap_Tabla_Prueba"
START TRANSACTION
SQL EXECUTE(SQLStmt;[Tabla 2]Campo1)
While(Not(SQL End selection))
   SQL LOAD RECORD
   ... `Escribir el código de validación de datos aquí
End while
VALIDATE TRANSACTION  `Validación de la transacción
```

Ejemplo 3

En este ejemplo, queremos obtener la columna ename de la tabla emp de la fuente de datos externos. El resultado se almacenará en un array aNombre. Obtenemos los registros de 10 en 10.

```
ARRAY STRING(30;aNombre;20)
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTE(SQLStmt;aNombre)
While(Not(SQL End selection))
   SQL LOAD RECORD(10)
End while
```

Ejemplo 4

En este ejemplo, queremos obtener las columnas ename y job de la tabla emp para un ID especifico ID (cláusula WHERE) de la fuente de datos externa. El resultado se almacena en las variables 4D vNombre y vJob. Sólo se recupera el primer registro.

```
SQLStmt:="SELECT ename, job FROM emp WHERE id = 3"
SQL EXECUTE(SQLStmt;vNombre;vJob)
SQL LOAD RECORD
```

Ejemplo 5

En este ejemplo, queremos obtener la columna Campo_Blob de la tabla Test en la fuente de datos. El resultado se almacena en una variable BLOB cuyo valor se actualiza cada vez que se carga un registro.

```
C_BLOB(MiBlob)
SQL LOGIN
SQL EXECUTE("SELECT Campo_Blob FROM Test";MiBlob)
While(Not(SQL End selection))
  `Buscamos en los resultados
   SQL LOAD RECORD
  `El valor de MiBlob se actualiza en cada llamada
```

Ejemplo 6

Usted quiere recuperar localmente los datos almacenados en una base 4D Server remota. Para hacerlo, deber utilizar arrays intermediarios:

```
 // Conexión a la base remota
SQL LOGIN("IP:192.168.18.15:19812";"user";"password";*)
If(OK=1)
 // A partir de este punto las peticiones se direccionan a la base remota
   C_TEXT($LastName_value) // variable 4D utilizada en la cadena de búsqueda
   ARRAY TEXT($a_LastName;0) // Almacenamiento temporal de los valores remotos de LastName
   ARRAY TEXT($a_FirstName;0) // Almacenamiento temporal de los valores remotos de FirstName
   C_BOOLEAN($UseSQL) //Elección del medio de almacenamiento de datos en local de la base remota
 // (demo only)

   $LastName_value:="Smith" // Inicialización de una variable 4D

 // Asociar la variable 4D $LastName_value variable con el primer "?" en la petición SQL
   SQL SET PARAMETER($LastName_value;SQL param in)

 // De la tabla remota PERSONS, recuperar los valores de los campos LastName y FirstName
 // donde "LastName = Smith" y almacenarlos en los arrays $a_LastName y $a_FirstName
   SQL EXECUTE("SELECT LastName, FirstName FROM PERSONS WHERE LastName = ?";$a_LastName;$a_FirstName)
   If(Not(SQL End selection)) // Si al menos se encuentra un registro

      SQL LOAD RECORD(SQL all records) // Cargar todos los registros

      $UseSQL:=True // Elija la forma de integrar los datos (demo únicamente)

      If($UseSQL) // Uso de las peticiones SQL
         SQL LOGOUT // Desconexión de la base remota
         SQL LOGIN(SQL_INTERNAL;"user";"password") // Conexión a la base local
 // A partir de este punto las peticiones SQL se direccionan a la base local
 // Guarde los arrays $a_LastName y $a_FirstName en la tabla local PERSONS
         SQL EXECUTE("INSERT INTO PERSONS(LastName, FirstName) VALUES (:$a_LastName, :$a_FirstName);")

      Else // Uso de comandos 4D
         For($i;1;Size of array($a_LastName))
            CREATE RECORD([PERSONS])
            [PERSONS]LastName:=$a_LastName{$i}
            [PERSONS]FirstName:=$a_FirstName{$i}
            SAVE RECORD([PERSONS])
         End for
      End if
   End if
   SQL LOGOUT // Cierre de la conexión
End if
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK devuelve 1, de lo contrario devuelve 0.

SQL EXECUTE SCRIPT

SQL EXECUTE SCRIPT (rutaScript ; accionError {; nomAtrib ; valAtrib} {; nomAtrib2 ; valAtrib2 ; ... ; nomAtribN ; valAtribN})

Parámetro	Tipo		Descripción
rutaScript	Texto	➡	Ruta de acceso completa del archivo que contiene el script SQL a ejecutar
accionError	Entero largo	➡	Acción a efectuar en caso de error durante la ejecución del script
nomAtrib	Texto	➡	Nombre del atributo a utilizar
valAtrib	Texto	➡	Valor del atributo

Descripción

*El comando **SQL EXECUTE SCRIPT** permite ejecutar una serie de instrucciones SQL ubicadas en el archivo de script designado por rutaScript.*
Este comando sólo puede ejecutarse en un equipo local (4D local o procedimiento almacenado en 4D Server). Funciona con la base de datos actual (base interna o externa)

***Nota:** este comando no puede utilizarse con una conexión externa que se abre directamente o vía ODBC.*

Pase en el parámetro rutaScript la ruta de acceso completa del archivo texto que contiene las instrucciones SQL a ejecutar. La ruta de acceso debe expresarse utilizando la sintaxis del sistema actual. Si pasa una cadena vacía ("") en rutaScript, se muestra una caja de diálogo de apertura estándar de manera que el usuario pueda seleccionar el archivo de script a ejecutar.

***Nota:** los comandos **SQL EXPORT DATABASE** y **SQL EXPORT SELECTION** generan automáticamente este archivo de script.*

El parámetro accionError se utiliza para configurar el funcionamiento del comando cuando ocurre un error durante la ejecución del script. Puede pasar una de las siguientes tres constantes, ubicadas en el tema "":

Constante	Tipo	Valor	Comentario
SQL On error abort	Entero largo	1	En caso de error, 4D detiene de inmediato la ejecución del script.
SQL On error confirm	Entero largo	2	En caso de error, 4D muestra una caja de diálogo que describe el error y permite al usuario interrumpir o continuar la ejecución del script.
SQL On error continue	Entero largo	3	En caso de error, 4D lo ignoras y continua la ejecución del script.

Los parámetros nomAtrib y valAtrib deben pasarse en pares. Estos parámetros permiten definir los atributos específicos para la ejecución del script. En la versión actual de 4D, en nomAtrib sólo puede pasarse un atributo, disponible vía la constante siguiente, ubicada en el tema "":

Constante	Tipo	Valor	Comentario
SQL use access rights	Cadena	SQL_Use_Access_Rights	Permite restringir los derechos de acceso a aplicar durante la ejecución de los comandos SQL del script. Cuando utilice este atributo, debe pasar 0 ó 1 en attribValue: • attribValue = 1: 4D utiliza los derechos de acceso del usuario 4D actual. • attribValue = 0 (o atributo no definido): 4D no restringe el acceso, se utilizan los derechos del Diseñador.

*Si el archivo de registro de peticiones de 4D está activo (vía los selectores 28 o 45 del comando **SET DATABASE PARAMETER**), cada comando SQL ejecutado generará una entrada con la siguiente información:*

- *Tipo de comando SQL*
- *Número de registros afectados por el comando*
- *Duración de ejecución del comando*
- *Para cada error encontrado:*
  - *el código de error*
  - *el texto del error si está disponible*

Si el script se ejecuta correctamente (no se encuentra ningún error), la variable sistema OK toma el valor 1. En el evento de un error, la variable sistema OK toma o no el valor 0 en función del parámetro accionError:

- *Si errorAction es SQL On error abort (valor 1), OK toma el valor 0.*
- *Si errorAction es SQL On error confirm (valor 2), OK toma el valor 0 si el usuario elije detener la operación y 1 si elije continuar.*
- *Si errorAction es SQL On error continue (valor 3), la variable OK siempre toma el valor 1.*

***Nota:** si utiliza este comando para ejecutar acciones consumidoras de memoria tales como importación masiva de datos, puede considerar llamar al comando SQL **ALTER DATABASE** para desactivar temporalmente las opciones SQL.*

SQL EXPORT DATABASE

SQL EXPORT DATABASE (rutaCarpeta {; numArchivos {; tamLimiteArchivos {; tamLimiteCampos}}})

Parámetro	Tipo		Descripción
rutaCarpeta	Texto	➡	Ruta de acceso de la carpeta de exportación o "" para mostrar una caja de diálogo de selección de carpeta
numArchivos	Entero largo	➡	Número máximo de archivos por carpeta
tamLimiteArchivos	Entero largo	➡	Valor límite de tamaño de los archivos de exportación (en KB)
tamLimiteCampos	Entero largo	➡	Limite de tamaño (en bytes) debajo del cual el contenido de un campo Texto, BLOB o Imagen se integrará al archivo principal

Descripción

*El comando **SQL EXPORT DATABASE** exporta al formato SQL todos los registros de todas las tablas de la base. En SQL, esta operación de exportación global se llama "Dump".*

***Nota:** este comando no puede utilizarse con una conexión externa abierta directamente o vía ODBC.*

*Para cada tabla, el comando genera un archivo de texto con las instrucciones SQL necesarias para la importación de los datos en otra base. Este archivo puede ser utilizado directamente por el comando **SQL EXECUTE SCRIPT** para importar los datos en otra base 4D.*

Los archivos de exportación se crearán en una carpeta llamada "SQLExport" ubicada en la carpeta de destino designada por el parámetro rutaCarpeta. Por favor tenga en cuenta que si la carpeta "SQLExport" ya existe en la ubicación especificada, el comando se reemplazará sin que se muestre ningún mensaje de advertencia.
Si pasa una cadena vacía en este parámetro, 4D muestra una caja de diálogo estándar permitiéndole al usuario designar la carpeta de destino. Por defecto, la caja de diálogo muestra la carpeta actual del usuario que abrió la sesión ("Mis Documentos" bajo Windows o "Documents" bajo Mac OS).

Para cada tabla exportada, el comando efectúa las siguientes acciones:

- *se crea una subcarpeta con el nombre de la tabla en la carpeta de destino.*
- *un archivo texto llamado "Export.sql" se crea en la subcarpeta. Este archivo está codificado en UTF-8 con BOM (marca de orden de bytes). Contiene las órdenes SQL **INSERT** correspondientes a los datos exportados. Los valores de los campos están separados por dos puntos. Puede tener menos valores que campos en la tabla. En este caso, los campos restantes se consideran NULOS.*
- *si la tabla contiene campos BLOB, imagen o texto (textos guardados externamente, en otras palabras, fuera de los registros), por defecto el comando crea una subcarpeta adicional llamada "BLOBS" junto al archivo "Export.sql" y crea tantas subcarpetas "BlobsX" como sea necesario. Estas subcarpetas guardarán como archivos separados el contenido de todos los campos BLOB, imagen o campos de texto externos de los registros de la tabla. Los archivos BLOB se llaman "BlobXXXXX.BLOB", los archivos texto se llaman "TEXTXXXXXX.TXT (donde XXXXX es un número único generado por la aplicación). Los archivos imagen se llaman PICTXXXXX.ZZZZ (donde XXXXX es un número único generado por la aplicación y ZZZZ es la extensión). Cuando es posible, las imágenes se exportan a su formato nativo de origen con la extensión correspondiente al tipo de imagen (.jpg, .png, etc.). Si la exportación no es posible en el formato nativo, las imágenes se exportan en el formato 4D interno con la extensión .4PCT.*

***Nota:** este funcionamiento difiere al ejecutar **SQL EXPORT DATABASE** desde un 4D en modo remoto. En este contexto, los datos a almacenar externamente se incluyen automáticamente en el archivo "Export.sql".*

Si pasa el parámetro numArchivos, el comando creará tantas subcarpetas "BlobsX" como sea necesario de manera que cada una de ellas no contenga más de numArchivos BLOB, imagen o textos externos. Por defecto, si se omite el parámetro numArchivos, el comando limita el número de archivos a 200. Si pasa 0, cada subcarpeta contendrá al menos un archivo.

El parámetro tamMaxArchivo permite definir un límite de tamaño (en KB) para cada archivo "Export.sql" creado en el disco. Cuando el tamaño del archivo de exportación alcanza el valor definido en tamMaxArchivo, 4D detiene la escritura de registros, cierra el archivo y crea un nuevo archivo llamado "ExportX.sql" (donde X representa el número de secuencia) junto al anterior. Note que este es un límite teórico: el tamaño actual de los archivos "ExportX.sql" supera el valor definido por tamMaxArchivo porque el archivo sólo se cierra después de que el registro que se estaba exportando cuando se alcanzó el límite se haya escrito completamente (los contenidos de los registros no se dividen). El tamaño mínimo aceptado es 100 KB y el valor máximo (valor por defecto) es 100 000 (10MB).

El parámetro opcional tamLimiteCampos permite definir una tamaño límite debajo del cual el contenido de un campo BLOB, Imagen, o texto externo se integrará al archivo principal "Export.sql" en lugar de guardarse como un archivo separado. El propósito de este parámetro es optimizar las operaciones de exportación limitando el número de subcarpetas y de archivos creados en el disco.
Este parámetro debe expresarse en bytes. Por ejemplo, si pasa 1000, todos los campos BLOB, imagen o texto externo que contienen los datos de un tamaño inferior o igual a 1000 bytes se integrarán al archivo de exportación principal.
*Note que los datos de los campos binarios (BLOB e Imagen) integrados al archivo de exportación se escriben en formato hexadecimal, de la forma X'0f20' (notación hexadecimal SQL estándar, ver **literal**). Este formato es soportado automáticamente por el motor SQL de 4D.*
Por defecto, si se omite el parámetro tamLimiteCampos, los campos BLOB, Imagen y texto externos siempre se exportan en forma de archivos externos, sin importar su tamaño.

En el archivo de exportación, puede haber menos valores que campos en la tabla. En este caso, los campos vacíos se considerarán como NULL. También puede pasar el valor NULL en un campo.

Si la exportación se realiza correctamente, la variable OK toma el valor 1. De lo contrario, toma el valor 0.

***Nota:** este comando no soporta campos de tipo Objeto.*

SQL EXPORT SELECTION

SQL EXPORT SELECTION (tabla ; rutaCarpeta {; numArchivos {; tamLimiteArchivos {; tamLimiteCampos}}})

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla de la cual exportar la selección
rutaCarpeta	Texto	➡	Ruta de acceso de la carpeta de exportación o "" para mostrar una caja de diálogo de selección de carpeta
numArchivos	Entero largo	➡	Número máximo de archivos por carpeta
tamLimiteArchivos	Entero largo	➡	Tamaño máximo del archivo Export.sql (en KB)
tamLimiteCampos	Entero largo	➡	Límite de tamaño debajo del cual el contenido de un campo Texto, BLOB o imagen se integrará al archivo principal (en bytes)

Descripción

*El comando **SQL EXPORT SELECTION** exporta al formato SQL los registros de la selección actual de la tabla 4D designada por el parámetro Tabla.*

*Este comando es casi idéntico al comando **SQL EXPORT DATABASE**. El archivo generado puede ser utilizado directamente por el comando [#cmd id="1089"/] con el fin de importar datos en otra base 4D. La única diferencia entre estos dos comandos es que **SQL EXPORT SELECTION** sólo exporta la selección actual de Tabla mientras **SQL EXPORT DATABASE** exporta la totalidad de datos de la base. Igualmente, a diferencia del comando **SQL EXPORT DATABASE**, este comando no funciona con las bases SQL externas. Sólo puede utilizarse con la base principal*

*Consulte la descripción del comando **SQL EXPORT DATABASE** para una descripción detallada del funcionamiento y parámetros de estos comandos.*

Si la selección actual está vacía, el comando no hace nada. Note que en este caso, la carpeta de destino no se vacía.

Si la exportación se lleva a cabo correctamente, la variable OK toma el valor 1. De lo contrario, toma el valor 0.

***Nota:** este comando no soporta campos de tipo Objeto.*

SQL GET LAST ERROR

SQL GET LAST ERROR (errCode ; errText ; errODBC ; errSQLServer)

Parámetro	Tipo		Descripción
errCode	Entero largo	⇐	Código del error
errText	Texto	⇐	Texto del error
errODBC	Texto	⇐	Código del error ODBC
errSQLServer	Entero largo	⇐	Código del error nativo servidor SQL

Descripción

*El comando **SQL GET LAST ERROR** devuelve la información relacionada con el último error encontrado durante la ejecución de un comando ODBC. El error puede venir de la aplicación 4D, la red, la fuente ODBC, etc.*

*Este comando generalmente debe llamarse en el contexto de un método de gestión de errores instalado utilizando el comando **ON ERR CALL**.*

- *El parámetro errCode devuelve el código del error.*
- *El parámetro errText devuelve el texto del error.*

Los dos últimos parámetros sólo se llenan cuando el error viene de la fuente ODBC; de lo contrario, se devuelven vacíos.

- *El parámetro errODBC devuelve el código del error ODBC (estado SQL).*
- *El parámetro errSQLServer devuelve el código del error nativo servidor SQL.*

SQL GET OPTION

SQL GET OPTION (opción ; valor)

Parámetro	Tipo		Descripción
opción	Entero largo	⇒	Número de opción
valor	Entero largo, Texto	⇐	Valor de la opción

Descripción

*El comando **SQL GET OPTION** devuelve el valor actual de la opción pasada en el parámetro opcion.*
*Para mayor información sobre las diferentes opciones y sus valores asociados, consulte la descripción del comando **SQL SET OPTION**.*

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0.

SQL LOAD RECORD

SQL LOAD RECORD {(numReg)}

Parámetro	Tipo		Descripción
numReg	Entero largo	➡	Número de registros a cargar

Descripción

El comando **SQL LOAD RECORD** recupera en 4D uno o más registros de la fuente de datos abierta en la conexión actual.
El parámetro opcional numReg se utiliza para definir el número de registros a recuperar:

- Si omite este parámetro, el comando recuperará el registro actual de la fuente de datos. Este principio corresponde a la recuperación de datos en un bucle donde se recibe un registro a la vez.
- Si pasa un valor entero en numReg, el comando recupera numReg registros.
- Si pasa la constante SQL All Records (valor -1), el comando recuperará todos los registros de la tabla.

Nota: estos dos últimos parámetros sólo tienen sentido si los datos recuperados están asociados con arrays o campos 4D.

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente, la variable sistema OK devuelve 1. De lo contrario, devuelve 0.

SQL LOGIN

SQL LOGIN {(entradaDatos ; nomUsuario ; contraseña ; *)}

Parámetro	Tipo		Descripción
entradaDatos	Cadena	➡	Nombre de publicación de base 4D o Dirección IP de la base remota o Nombre de la fuente de datos en el administrador ODBC o "" para mostrar el diálogo de selección
nomUsuario	Cadena	➡	Nombre del usuario registrado en la fuente de datos
contraseña	Cadena	➡	Contraseña del usuario registrado en la fuente de datos
*	Operador	➡	Aplicado a Begin SQL/End SQL Si se omite: no aplicar (base de datos local); si se pasa: aplicar

Descripción

*El comando **SQL LOGIN** permite conectarse a una fuente de datos SQL especificada en el parámetro entradaDatos y designa el objetivo de las búsquedas SQL ejecutadas posteriormente en el proceso actual:*

- *vía el comando **SQL EXECUTE**,*
- *vía el código ubicado dentro de las etiquetas Begin SQL / End SQL (si se pasa el parámetro *).*

La fuente de datos SQL puede ser:

- *una base 4D Server externa a la que acceda directamente,*
- *una fuente ODBC externa,*
- *la base 4D local (base interna).*

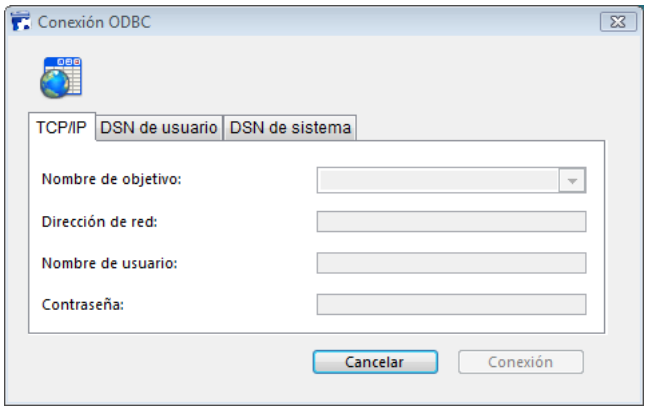
En entradaDatos, puede pasar uno de los siguientes valores: una dirección IP, un nombre de publicación de base 4D, un nombre de fuente de datos ODBC, una cadena vacía o la constante SQL_INTERNAL.

- ***Dirección IP***
  *Sintaxis: **IP:<Dirección IP>{:<Puerto TCP>}***
  En este caso, el comando abre una conexión directa con la base 4D Server ejecutada en la máquina con la dirección IP definida. En la máquina "objetivo", debe iniciarse el servidor SQL. Si pasa un número de puerto TCP, debe haber sido especificado como puerto de publicación del servidor SQL en la base "objetivo". Si no pasa el número de puerto TCP, se utilizará el número de puerto por defecto (19812). El número de puerto TCP del servidor SQL puede modificarse en la página "SQL" de las Propiedades de la base. Consulte los ejemplos 4 y 5.
  Si activa el TLS para el servidor SQL "objetivo" (opción disponible en las Propiedades de la base), debe añadir la palabra clave ":ssl" al final de la dirección IP y el número de puerto TCP (obligatorio en ese caso) para que el servidor pueda manejar la petición correctamente (ver el ejemplo 6).

- ***Nombre de la publicación de base 4D***
  Sintaxis: 4D:<Nombre_de_Publicación>
  En este caso, el comando abre una conexión directa con la base 4D Server cuyo nombre de publicación en la red corresponde al nombre especificado. El nombre de la publicación de red de una base se define en la página "Cliente-Servidor/Configuración" de las Propiedades de la base.
  Consulte el ejemplo 4.
  ***Nota:** el número de puerto TCP del servidor SQL 4D objetivo (que publica la base 4D) y el número de puerto TCP del servidor SQL de la aplicación 4D que abre la conexión deben ser idénticos.*

- ***nombre de fuente de datos ODBC válida***
  *Sintaxis: **ODBC:<Mi_DSN> o <Mi_DSN>***
  En este caso, el parámetro entradaDatos contiene el nombre de la fuente de datos como ha sido definida en el administrador del driver ODBC.
  Notas:
  - *Por razones de compatibilidad con las versiones anteriores de 4D, es posible omitir el prefijo "ODBC:". Sin embargo, por razones de legibilidad del código, se recomienda utilizar este prefijo. Consulte el ejemplo 2.*
  - *En Windows, el nombre de la fuente de datos distingue entre mayúsculas y minúsculas. Por ejemplo, si la fuente de datos se definió como "4D_v16", pasar el valor "4D_V16" fallará.*
  - *En Windows y Mac, el prefijo "ODBC:" debe introducirse con letras mayúsculas. Si pasa "odbc:", la conexión fallará.*

- ***cadena vacía***
  Sintaxis: ""
  En este caso, el comando muestra la caja de diálogo de conexión de manera que la fuente de datos a conectar pueda introducirse manualmente:



Esta caja de diálogo incluye varias páginas. La página TCP/IP incluye los siguientes elementos:

- *Nombre de objetivo: este menú se crea utilizando dos listas:*
  - *la lista de bases abiertas recientemente en conexión directa. El mecanismo para actualizar esta lista es idéntico al de la aplicación 4D, excepto que la carpeta que contiene los archivos .4DLink se llama "Favorites SQL vXX" en lugar de "Favorites vXX".*
  - *la lista de aplicaciones 4D Server cuyo servidor SQL se inicia y cuyo puerto TCP para las conexiones SQL es el mismo que para la aplicación fuente. Esta lista se actualiza dinámicamente en cada nueva llamada al comando **SQL LOGIN** sin el parámetro entradaDatos. Si el carácter "^" se ubica antes de un nombre de la base, indica que la conexión se efectúo en modo seguro vía TLS.*
- *Dirección de red: esta área muestra la dirección y posiblemente el puerto TCP de la base seleccionada en el menú Nombre de objetivo. También puede introducir una dirección IP en esta área y luego hacer clic en el botón de conexión para conectar la base 4D Server correspondiente. También puede especificar el puerto TCP introduciendo dos puntos (:) seguidos por el número de puerto después de la dirección. Por ejemplo: 192.168.93.105:19855*
- *Nombre de usuario y Contraseña: estas áreas pueden utilizarse para introducir los identificadores de conexión.*
- *Las páginas DSN de usuario y DSN de sistema muestran, respectivamente, la lista de usuario y fuentes de datos ODBC del sistema en el driver ODBC de la máquina. Estas páginas pueden utilizarse para seleccionar una fuente de datos e introducir identificadores para abrir una conexión con una fuente de datos externa ODBC.*

*Si se establece la conexión, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0 y se genera un error. Este error puede interceptarse vía un método de gestión de errores instalado por el comando **ON ERR CALL**.*

- ***Constante SQL_INTERNAL***
  Sintaxis: SQL_INTERNAL
  En este caso, el comando redirecciona las búsquedas SQL posteriores a la base 4D interna.

***Atención:** los prefijos utilizados en el parámetro entradaDatos (IP, ODBC, 4D) deben escribirse en mayúsculas.*

nombreUsuario contiene el nombre del usuario autorizado a conectarse a la fuente de datos externa. Por ejemplo, con Oracle®, el nombre de usuario puede ser "Samuel".

contraseña contiene la contraseña del usuario autorizado a conectarse a la fuente de datos externos. Por ejemplo, conOracle®, la contraseña puede ser "tiger".
***Nota:** en el caso de una conexión directa, si pasa una cadena vacía en los parámetros nombreUsuario y contraseña, la conexión sólo se aceptará si las contraseñas 4D no están activas en la base objetivo. De lo contrario, la conexión se rechazará.*

*El parámetro opcional * puede utilizarse para cambiar el objetivo del código SQL ejecutado dentro de las etiquetas Begin SQL/End SQL. Si no pasa este parámetro, el código ubicado dentro de las etiquetas Begin SQL/End SQL aún se enviará al motor SQL interno de 4D, sin tener en cuenta la configuración especificada por el comando **SQL LOGIN** . Si pasa este parámetro, el código SQL ejecutado dentro de las etiquetas Begin SQL/End SQL se enviará a la fuente especificada en el parámetro entradaDatos.*

*Para cerrar la conexión actual y liberar la memoria, simplemente ejecute el comando **SQL LOGOUT**. Todas las búsquedas SQL se envían a la base 4D SQL interna.*
*Si llama nuevamente a **SQL LOGIN** sin haber cerrado explícitamente la conexión actual, la conexión se cerrará automáticamente.*

***Nota:** en caso de falla de un intento de conexión externa vía **SQL LOGIN**, la base 4D interna se convierte automáticamente en la fuente de datos actual.*

Estos parámetros son opcionales; si no se pasa ningún parámetro, el comando produce la visualización de la caja de diálogo Seleccionar origen de datos ODBC, que le permite seleccionar la fuente de datos externos:

El alcance de este comando es el proceso; en otras palabras, si quiere ejecutar dos conexiones distintas, debe crear dos procesos y ejecutar cada conexión en cada proceso.

***Atención:** no es posible abrir una conexión ODBC en el contexto descrito abajo. Estas configuraciones conllevan al bloqueo de la aplicación:*

- *conexión vía ODBC desde la aplicación en ejecución hacia ella misma*
- *conexión vía ODBC desde una aplicación 4D a 4D Server cuando una conexión cliente/servidor clásica ya está abierta entre estas dos aplicaciones.*

Ejemplo 1

Esta instrucción provoca la visualización de la caja de diálogo Seleccionar origen de datos ODBC:

```
SQL LOGIN
```

Ejemplo 2

*Apertura de una conexión vía el protocolo ODBC con la fuente de datos externa "MyOracle". Las búsquedas SQL ejecutadas vía el comando **SQL EXECUTE** y búsquedas incluidas dentro de las etiquetas **Begin SQL**/**End SQL** se redireccionará para esta conexión. Esta instrucción conectará la fuente de datos ODBC llamada "MyOracle" utilizando Scott/tiger como nombre/contraseña:*

```
SQL LOGIN("ODBC:MyOracle";"Scott";"tiger";*)
```

Ejemplo 3

Apertura de una conexión con el motor SQL interno de 4D:

```
SQL LOGIN(SQL_INTERNAL;$user;$password)
```

Ejemplo 4

*Apertura de una conexión directa con la aplicación 4D Server ejecutada en la máquina con la dirección IP 192.168.45.34 y respondiendo en el puerto TCP por defecto. Las búsquedas SQL ejecutadas vía el comando **SQL EXECUTE** se redireccionan a esta conexión; las búsquedas incluidas dentro de las etiquetas **Begin SQL**/**End SQL** no se redireccionarán.*

```
SQL LOGIN("IP:192.168.45.34";"Juan";"azerty")
```

Ejemplo 5

*Apertura de una conexión directa con la aplicación 4D Server ejecutada en la máquina con la dirección IP 192.168.45.34 y respondiendo al puerto TCP 20150. Las búsquedas SQL ejecutadas vía el comando **SQL EXECUTE** y las búsquedas incluidas dentro de las etiquetas **Begin SQL**/**End SQL** se redireccionarán a esta conexión.*

```
SQL LOGIN("IP:192.168.45.34:20150";"Juan";"azerty";*)
```

Ejemplo 6

Apertura de una conexión directa en SSL con la aplicación 4D Server ejecutada en la máquina con la dirección IP 192.168.45.34 y responder en el puerto TCP por defecto. Debe tener activado SSL para el servidor SQL en la aplicación 4D Server:

```
SQL LOGIN("IP:192.168.45.34:19812:ssl";"Admin";"sd156") // Note the ":ssl" after of the IP address and TCP port
```

Ejemplo 7

*Apertura de una conexión directa con la aplicación 4D Server ejecutada en la máquina que tiene la dirección IPv6 2a01:e35:2e41:c960:dc39:3eb0:f29b:3747 y responde en el puerto TCP 20150. Las consultas SQL ejecutadas a través del comando **SQL EXECUTE** serán redirigidas a esta conexión; Las consultas incluidas en las etiquetas **Begin SQL**/**End SQL** no se redirigirán.*

```
SQL LOGIN("IP:[2a01:e35:2e41:c960:dc39:3eb0:f29b:3747]:20150";"John";"qwerty")
```

Ejemplo 8

*Apertura de una conexión directa con la aplicación 4D Server que publica en la red local una base cuyo nombre de publicación es "Accounts_DB." El puerto TCP utilizado por el servidor SQL de ambas bases (definido en la página "SQL" de las Propiedades de la base) debe ser el mismo (19812 por defecto). Las búsquedas SQL ejecutadas vía el comando **SQL EXECUTE** se redireccionarán a esta conexión; las búsquedas incluidas dentro de las etiquetas **Begin SQL**/**End SQL** no se redireccionarán.*

```
SQL LOGIN("4D:Accounts_DB";"Juan";"azerty")
```

Ejemplo 9

*Este ejemplo muestra las posibilidades de conexión ofrecidas por el comando **SQL LOGIN**:*

```
ARRAY TEXT(aNombres;0)
ARRAY LONGINT(aEdades;0)SQL LOGIN("ODBC:MyORACLE";"Marc";"azerty")
If(OK=1) `La siguiente búsqueda se redireccionará a la base de datos ORACLE externa
   SQL EXECUTE("SELECT Nombre, edad FROM PERSONS";aNombres;aEdades)
 `La siguiente búsqueda se enviará a la base de datos 4D local
   Begin SQL
      SELECT Nombre, Edad
```

```
        FROM PERSONS
        INTO :aNombres, :aEdades;
    End SQL  `El siguiente comando SQL LOGIN cierra la conexión actual
  `con la base de datos ORACLE externa y abre una nueva conexión
  `con una base de datos MySQL externa
    SQL LOGIN("ODBC:MySQL";"Juan";"qwerty";*)
    If(OK=1)
  `La siguiente búsqueda será redireccionada a la base de datos MySQL externa
        SQL EXECUTE("SELECT Nombre, Edad FROM PERSONS";aNombres;aEdades)
  `La siguiente búsqueda también será redireccionada a la base de datos MySQL externa
        Begin SQL
            SELECT Nombre, Edad
            FROM PERSONS
            INTO :aNombres, :aEdades;
        End SQL
        SQL LOGOUT
  `La siguiente búsqueda se enviará a la base de datos 4D local
        Begin SQL
            SELECT Nombre, Edad
            FROM PERSONS
            INTO :aNombres, :aEdades;
        End SQL
    End if
End if
```

Variables y conjuntos del sistema

Si la conexión es exitosa, el variable sistema OK toma el valor 1; de lo contrario, toma el valor 0.

SQL LOGOUT

SQL LOGOUT
Este comando no requiere parámetros

Descripción

*El comando **SQL LOGOUT** cierra la conexión con una fuente ODBC abierta en el proceso actual (si aplica). Si no hay conexión ODBC abierta, el comando no hace nada.*

Variables y conjuntos del sistema

*Si la conexión se cierra correctamente, la variable sistema OK toma el valor 1; de lo contrario, toma el valor 0. Puede interceptar este error con la ayuda de un método instalado por el comando **ON ERR CALL**.*

SQL SET OPTION

SQL SET OPTION (opción ; valor)

Parámetro	Tipo		Descripción
opción	Entero largo	➡	Número de opción a definir
valor	Entero largo, Cadena	➡	Nuevo valor de opción

Descripción

*El comando **SQL SET OPTION** se utiliza para modificar el valor de la opción pasada en opcion.*
opcion puede tener uno de los siguientes valores, ubicados en el tema "SQL":

Constante	Tipo	Valor	Comentario
SQL asynchronous	Entero largo	1	0 = Conexión sincrónica (valor por defecto), 1 (o valor diferente de 0) = Conexión asincrónica
SQL charset	Entero largo	100	Codificación del texto utilizada por las peticiones enviadas a fuentes externas (vía SQL pass-through). La modificación se lleva a cabo para el proceso actual y la conexión actual. Valores posibles: identificador MIBEnum (ver nota 2), cadena "WCHAR" (ver nota 3) o valor -2 (ver nota 4). Por defecto: 106 (UTF-8)
SQL connection timeout	Entero largo	5	Tiempo máximo de espera durante la ejecución del comando **SQL LOGIN**. Este valor debe definirse antes de abrir la conexión para que sea tenido en cuenta. Valores posibles: tiempo en segundos Por defecto: no hay timeout
SQL max data length	Entero largo	3	Longitud máxima de los datos devueltos
SQL max rows	Entero largo	2	Número máximo de líneas en el conjunto resultante (utilizado para previsualizaciones)
SQL query timeout	Entero largo	4	Tiempo máximo de espera al ejecutar el comando **SQL EXECUTE**. Valores: tiempo en segundos Por defecto: no hay timeout

Notas:

*1. Cuando trabaja con el motor SQL interno de 4D, la opción **SQL Asynchronous** no tienen ningún propósito ya que ese tipo de conexión siempre es sincrónica.*
2. Los números MIBEnum se referencian en las siguientes direcciones: http://www.iana.org/assignments/character-sets.
*3. Cuando pasa -2 como el valor a **SQL Charset**, la codificación utilizada por el servidor SQL de 4D se adapta automáticamente a la plataforma de ejecución (codificación no UTF):*

- *Bajo Windows, se utiliza ISO8859-1,*
- *Bajo Mac OS, se utiliza MAC-ROMAN.*

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK devuelve 1. De lo contrario, devuelve 0.

SQL SET PARAMETER

SQL SET PARAMETER (objeto ; tipoParam)

Parámetro	Tipo		Descripción
objeto	Objeto 4D	➡	Objeto 4D a utilizar (variable, array o campo)
tipoParam	Entero largo	➡	Tipo de parámetro

Descripción

*El comando **SQL SET PARAMETER** permite el uso del valor de una variable, array o campo 4D en las peticiones SQL.*

***Nota:** igualmente es posible insertar directamente el nombre de un objeto 4D a utilizar (variable, array o campo) entre los caracteres << y >> en el texto de la petición (ver ejemplo 1). Para mayor información, consulte la sección **Presentación de los comandos del tema SQL**.*

- *En el parámetro objeto, pase el objeto 4D (variable, array o campo) a utilizar en la petición.*
- *En el parámetro tipoParam, pase el tipo SQL del parámetro. Puede pasar un valor o utilizar una de las siguientes constantes, ubicadas en el tema "**SQL**":*

Constante	Tipo	Valor	Comentario
SQL param in	Entero largo	1	
SQL param in out	Entero largo	2	Utilizable únicamente en el contexto de un procedimiento almacenado SQL (entrada-salida parámetro definido en el procedimiento almacenado)
SQL param out	Entero largo	4	Utilizable únicamente en el contexto de un procedimiento almacenado SQL (parámeto salida definido en el procedimiento almacenado)

*El valor del objeto 4D reemplaza al carácter ? en la petición SQL (sintaxis estándar). Si la petición contiene más de un carácter ?, serán necesarias varias llamadas a **SQL SET PARAMETER**. Los valores de los objetos 4D serán asignados secuencialmente en la petición, en el orden de ejecución de los comandos.*

***Atención:** este comando se utiliza para manejar los parámetros pasados a la petición SQL. No es posible utilizar el tipo SQL param out para asociar un objeto 4D al resultado de una petición SQL. El resultado de una petición SQL se recupera, por ejemplo, utilizando el parámetro objAsoc del comando **SQL EXECUTE** (ver **Presentación de los comandos del tema SQL**). El comando **SQL SET PARAMETER** se utiliza por lo general para definir los parámetros pasados a la petición (SQL param in); los tipos SQL param out y SQL param in out están reservados para ser utilizados en el contexto de los procedimientos almacenados SQL que pueden devolver parámetros.*

Ejemplo 1

Este ejemplo se utiliza para ejecutar una petición SQL la cual llama directamente a las variables 4D asociadas:

```
C_TEXT(MiTexto)
C_LONGINT(MiEnteroLargo)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<MiTexto>>, <<MienteroLargo>>)"
For(vContador;1;10)
   MiTexto:="Text"+String(vContador)
   MiEnteroLargo:=vContador
   SQL EXECUTE(SQLStmt)
   SQL CANCEL LOAD
End for
SQL LOGOUT
```

Ejemplo 2

*El mismo ejemplo anterior, pero utilizando el comando **SQL SET PARAMETER**:*

```
C_TEXT(MiTexto)
C_LONGINT(MiEnteroLargo)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (?,?)"
For(vCounter;1;10)
   MiTexto:="Text"+String(vContador)
   MiEnteroLargo:=vCounter
   SQL SET PARAMETER(MyText;SQL param in)
   SQL SET PARAMETER(MyLongint;SQL param in)
   SQL EXECUTE(SQLStmt)
   SQL CANCEL LOAD
```

```
End for
SQL LOGOUT
```

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente, la variable sistema OK devuelve 1. De lo contrario, devuelve 0.

START SQL SERVER

START SQL SERVER
Este comando no requiere parámetros

Descripción

*El comando **START SQL SERVER** lanza el servidor SQL integrado de la aplicación 4D en la que se ejecuta. Una vez lanzado, el servidor SQL puede responder a las búsquedas SQL externas.*

***Nota:** este comando no afecta el funcionamiento del motor SQL interno de 4D. El motor SQL siempre está disponible para las búsquedas internas.*

Variables y conjuntos del sistema

Si el servidor SQL ha sido lanzado correctamente, la variable sistema OK toma el valor, de lo contrario toma el valor 0.

STOP SQL SERVER

STOP SQL SERVER
Este comando no requiere parámetros

Descripción

*El comando **STOP SQL SERVER** detiene el servidor SQL integrado de la aplicación 4D en la cual ha sido ejecutado.*

Si el servidor SQL ha sido lanzado, todas las conexiones SQL se interrumpen y el servidor no acepta más búsquedas SQL externas. Si el servidor SQL no se lanzó, el comando no hace nada.

***Nota:** este comando no afecta el funcionamiento del motor SQL interno de 4D. El motor SQL siempre está disponible para búsquedas internas.*

_o_USE EXTERNAL DATABASE

_o_USE EXTERNAL DATABASE (nomFuente {; usuario ; contraseña})

Parámetro	Tipo		Descripción
nomFuente	Cadena	➡	Nombre de la fuente de datos ODBC a la cual conectarse
usuario	Cadena	➡	Nombre del usuario
contraseña	Cadena	➡	Contraseña del usuario

Nota de compatibilidad

*Este comando fue reemplazado por el comando **SQL LOGIN** desde la versión 11.3 de 4D. No debe utilizarse en sus desarrollos.*

_o_USE INTERNAL DATABASE

_o_USE INTERNAL DATABASE

Este comando no requiere parámetros

Nota de compatibilidad

*Este comando fue reemplazado por el comando **SQL LOGOUT** a partir de la versión 11.3 de 4D. No debe utilizarse más en sus desarrollos.*

Subregistros

- *Get subrecord key*
- *_o_ALL SUBRECORDS*
- *_o_APPLY TO SUBSELECTION*
- *_o_Before subselection*
- *_o_CREATE SUBRECORD*
- *_o_DELETE SUBRECORD*
- *_o_End subselection*
- *_o_FIRST SUBRECORD*
- *_o_LAST SUBRECORD*
- *_o_NEXT SUBRECORD*
- *_o_ORDER SUBRECORDS BY*
- *_o_PREVIOUS SUBRECORD*
- *_o_QUERY SUBRECORDS*
- *_o_Records in subselection*

Get subrecord key

Get subrecord key (campoID) -> Resultado

Parámetro	Tipo		Descripción
campoID	Campo	➡	Campo de tipo "Relación subtabla" o "Entero largo" de una relación subtabla anterior
Resultado	Entero largo	➲	Llave interna de la relación

Descripción

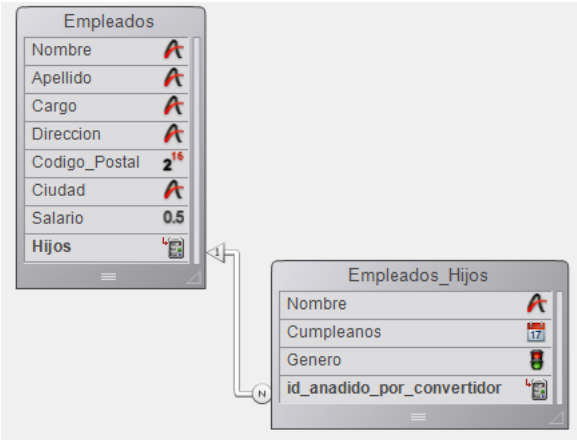
*El comando **Get subrecord key** facilita la migración del código 4D utilizando subtablas convertidas al código estándar de manipulación de tablas.*

***Recordatorio:** a partir de la versión 11 de 4D, no se soportan subtablas. Cuando se convierte una base antigua, las subtablas existentes se transforman automáticamente en tablas estándar relacionadas con las tablas originales por una relación automática. La subtabla anterior se convierte en la tabla Muchos y la tabla original es la tabla Uno. En la tabla Uno, el campo subtabla anterior se transforma en un campo especial de tipo "Relación subtabla" y en la tabla Muchos, se añade un campo especial, de tipo "Relación subtabla", llamado "id_anadido_por_convertidor".*

Esto permite preservar el funcionamiento de bases de datos convertidas, pero le recomendamos que sustituya los mecanismos de subtablas en sus bases por los que se utilizan para las tablas estándar.

*El primer paso en este proceso consiste en eliminar las relaciones automáticas especiales, lo que desactiva de forma permanente los mecanismos heredados de subtablas. Después debe volver a escribir el código asociado. El comando **Get subrecord key** acompaña a esta reescritura, devolviendo el identificador interno utilizado por la relación. Esta identificación interna hace que la relación real innecesaria y a continuación puede trabajar con la selección de la subtabla anterior incluso cuando la relación ya no está presente.*

Veamos por ejemplo la siguiente estructura convertida:



En 4D, el siguiente código aún funciona, pero debe actualizarse:

```
ALL SUBRECORDS([Empleados]Hijos)
$total:=Records in subselection([Empleados]Hijos)
vNombres:=""
For($i;1;$total)
   vNombre:=vNombres+[Empleados]Hijos'Nombre+" "
   NEXT SUBRECORD([Empleados]Hijos)
End for
```

Ahora puede reemplazar este código por:

```
QUERY([Empleados_Hijos];[Empleados_Hijos]id_anadido_por_convertidor=Get subrecord key([Empleados]Hijos))
$total:=Records in selection([Empleados_Hijos])
vNombres:=""
For($i;1;$total)
   vNombres:=vNombres+[Empleados_Hijos]Nombre+" "
   NEXT RECORD([Empleados_Hijos])
End for
```

***Nota:** **Get subrecord key** devuelve 0 si no hay registro actual cargado durante su ejecución.*

La segunda parte de código tiene la ventaja de usar comandos estándar de 4D y funciona de la misma manera si la relación está presente o no. Cuando elimina la relación, el comando devuelve simplemente el valor de la llave almacenada en el campo Entero largo.

En el parámetro campoID, el comando acepta un campo de tipo Relación subtabla (si la relación sigue existiendo) o del tipo Entero largo (si la relación se ha eliminado). En todos los demás casos, se genera un error.

Esto le permite escribir el código de transición. Durante la fase final de actualización de la aplicación, puede eliminar las llamadas a este comando.

Asignar el campo id_added_by_converter

*A partir de 4D v14 R3, puede asignar un valor al campo "id_added_by_converter". Anteriormente, este valor sólo podía ser asignado por 4D, lo que obligaba a los desarrolladores a utilizar comandos obsoletos como **_o_CREATE SUBRECORD** para poder añadir registros en subtablas convertidas.*

Gracias a esta posibilidad, puede convertir bases anteriores que contengan subtablas progresivamente: en un primer momento, puede mantener la "relación Subtabla" especial, mientras añade o modifica registros relacionados, como si fueran estándar. Luego, una vez que todos sus métodos estén actualizados, puede sustituir esta relación especial con una normal, sin tener que cambiar el código.

Por ejemplo, con la estructura anterior puede escribir:

```
CREATE RECORD([Employees])
[Employees]LastName:="Jones"
CREATE RECORD([Employees_Children])
[Employees_Children]FirstName:="Natacha"
[Employees_Children]BirthDate:=!12/24/2013!
[Employees_Children]id_added_by_converter:=Get subrecord key([Employees]Children)
SAVE RECORD([Employees_Children])
SAVE RECORD([Employees]
```

Este código funcionará tanto con una relación especial como con una estándar.

_o_ALL SUBRECORDS

_o_ALL SUBRECORDS (subtabla)

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla en la cual seleccionar los subregistros

Nota de compatibilidad

A partir de la versión 11 de 4D no se soportan subtablas. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, se recomienda remplazar las subtablas con tablas relacionadas estándar.

_o_APPLY TO SUBSELECTION

_o_APPLY TO SUBSELECTION (subtabla ; formula)

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla a la cual aplicar la fórmula
formula	Instrucción	➡	Línea de código o método

Nota de compatibilidad

Las subtablas no se soportan a partir de la versión 11 de 4D. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, es recomendable remplazar las subtablas con tablas relacionadas estándar.

_o_Before subselection

_o_Before subselection (subtabla) -> Resultado

Parámetro	Tipo		Descripción
subtabla	Subtabla	→	Subtabla para la cual probar si el puntero se encuentra antes del primer subregistro seleccionado
Resultado	Booleano	←	Sí (TRUE) o No (FALSE)

Nota de compatibilidad

Las subtablas no son soportadas a partir de la versión 11 de 4D. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, es recomendable remplazar las subtablas con tablas relacionadas estándar.

_o_CREATE SUBRECORD

_o_CREATE SUBRECORD (subtabla)

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla en la cual crear un nuevo subregistro

Nota de compatibilidad

Las subtablas no se soportan a partir de la versión 11 de 4D. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, es recomendable remplazar las subtablas con tablas relacionadas estándar.

_o_DELETE SUBRECORD

_o_DELETE SUBRECORD (subtabla)

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla de la cual borrar el subregistro actual

Nota de compatibilidad

A partir de la versión 11 de 4D no se soportan subtablas. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, se recomienda remplazar las subtablas con tablas relacionadas estándar.

_o_End subselection

_o_End subselection (subtabla) -> Resultado

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla para la cual probar si el puntero del subregistro actual está después del último subregistro seleccionado
Resultado	Booleano	⮌	Sí (TRUE) o No (FALSE)

Nota de compatibilidad

Las subtablas no se soportan a partir de la versión 11 de 4D. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, se recomienda remplazar las subtablas con tablas relacionadas estándar.

_o_FIRST SUBRECORD

_o_FIRST SUBRECORD (subtabla)

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla de la cual cargar el primer subregistro de la selección actual

Nota de compatibilidad

Las subtablas no se soportan a partir de la versión 11 de 4D. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, es recomendable remplazar las subtablas con tablas relacionadas estándar.

_o_LAST SUBRECORD

_o_LAST SUBRECORD (subtabla)

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla en la cual ubicar el último subregistro seleccionado

Nota de compatibilidad

Las subtablas no son soportadas a partir de la versión 11 de 4D. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, es recomendable remplazar las subtablas con tablas relacionadas estándar.

_o_NEXT SUBRECORD

_o_NEXT SUBRECORD (subtabla)

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla en la cual moverse al siguiente subregistro seleccionado

Nota de compatibilidad

Las subtablas no se soportan a partir de la versión 11 de 4D. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, es recomendable remplazar las subtablas con tablas relacionadas estándar.

_o_ORDER SUBRECORDS BY

_o_ORDER SUBRECORDS BY (subtabla ; subCampo {; > o <}{; subCampo2 ; > o <2 ; ... ; subCampoN ; > o <N})

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla que contiene los subregistros a ordenar
subCampo	Subcampo	➡	Subcampo en el cual efectuar la ordenación
> o <	Operador	➡	Dirección de la ordenación para cada nivel: > orden ascendente o < orden descendente

Nota de compatibilidad

A partir de la versión 11 de 4D no se soportan subtablas. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, se recomienda remplazar las subtablas con tablas relacionadas estándar.

_o_PREVIOUS SUBRECORD

_o_PREVIOUS SUBRECORD (subtabla)

Parámetro	*Tipo*		*Descripción*
subtabla	*Subtabla*	➡	*Subtabla en la cual moverse al subregistro selecionado previamente*

Nota de compatibilidad

A partir de la versión 11 de 4D no se soportan subtablas. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, se recomienda remplazar las subtablas con tablas relacionadas estándar.

_o_QUERY SUBRECORDS

_o_QUERY SUBRECORDS (subtabla ; formula)

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla en la cual realizar la búsqueda
formula	Booleano	➡	Fórmula de búsqueda

Nota de compatibilidad

A partir de la versión 11 de 4D no se soportan subtablas. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, se recomienda remplazar las subtablas con tablas relacionadas estándar.

_o_Records in subselection

_o_Records in subselection (subtabla) -> Resultado

Parámetro	Tipo		Descripción
subtabla	Subtabla	➡	Subtabla en la cual contar el número de subregistros
Resultado	Entero largo	⮌	Número de subregistros en la subselección actual

Nota de compatibilidad

Las subtablas no se soportan a partir de la versión 11 de 4D. Un mecanismo de compatibilidad asegura el funcionamiento de este comando en bases compartidas; sin embargo, se recomienda remplazar las subtablas con tablas relacionadas estándar.

SVG

- *Presentación de los comandos SVG*
- *SVG EXPORT TO PICTURE*
- *SVG Find element ID by coordinates*
- *SVG Find element IDs by rect*
- *SVG GET ATTRIBUTE*
- *SVG SET ATTRIBUTE*
- *SVG SHOW ELEMENT*

Presentación de los comandos SVG

SVG (Scalable Vector Graphics) es un formato de archivo basado en XML (extensión.svg) que se utiliza para describir gráficos vectoriales. SVG es más comúnmente utilizado para la publicación de datos estadísticos o cartográficos.

Estos archivos pueden ser vistos en los navegadores web, ya sea de forma nativa o mediante plug-ins. 4D incluye un motor de renderización SVG que le permite visualizar archivos SVG en los campos o variables imagen. El comando ***SVG EXPORT TO PICTURE*** *permite generar una imagen en 4D a partir de una descripción SVG. También tenga en cuenta que el comando* ***GRAPH*** *hace uso del motor SVG integrado de 4D.*

Para obtener más información sobre este formato, consulte la siguiente dirección: http://www.w3.org/Graphics/SVG/.

SVG EXPORT TO PICTURE

SVG EXPORT TO PICTURE (elementRef ; varImag {; tipoExport})

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML raíz
varImag	Imagen	➡	Variable imagen a recibir del árbol XML (imagen SVG)
tipoExport	Entero largo	➡	0 = No guardar la fuente de datos 1 = Copiar la fuente de datos 2 (por defecto) = Fuente de datos propia

Descripción

*El comando **SVG EXPORT TO PICTURE** permite guardar en la variable o campo imagen indicado por el parámetro **VarImag** una imagen en formato SVG contenida en un árbol XML.*

***Nota:** para mayor información sobre el formato SVG, consulte la sección .*

SVG (Scalable Vector Graphics) es un formato de archivo utilizado para describir en XML gráficos vectoriales (extension .svg). Estos archivos pueden ser visualizados en los navegadores web nativamente, o vía plug-ins. 4D v11 incluye un motor de renderización SVG que permite visualizar los archivos SVG en los campos o variables imagen. El uso más común de SVG es la publicación de datos estadísticos o cartográficos. Para mayor información sobre este formato, consulte la dirección: http://www.w3.org/Graphics/SVG/.

Pase en refElement la referencia del elemento XML raíz que contiene la imagen SVG.

Pase en varImag el nombre de la variable imagen o del campo imagen 4D que contendrá la imagen SVG. La imagen se exporta en su formato nativo (descripción XML) y es redibujada vía el motor de renderización SVG en el momento de la visualización.

*El parámetro opcional tipoExport permite definir la manera cómo la fuente de datos XML debe ser manejada por el comando. Puede pasar una de las siguientes constantes, que se encuentran en el tema "**XML**", en este parámetro:*

- ***Get XML Data Source** (0): 4D lee únicamente la fuente de datos XML; no se conserva con la imagen. Este parámetro acelera notablemente la velocidad de ejecución del comando; sin embargo, como el árbol DOM no se conserva, no es posible guardar ni exportar la imagen.*
- ***Copy XML Data Source** (1): 4D conserva una copia del árbol DOM con la imagen, lo que significa que la imagen puede guardarse en un campo de la base de datos y en cualquier momento mostrarla nuevamente o exportarla.*
- ***Own XML Data Source** (2): 4D exporta el árbol DOM con la imagen. La imagen puede almacenarse o exportarse y la ejecución del comando es rápida. Sin embargo, la referencia XML refElement no puede ser utilizada más por los otros comandos 4D. Este es el modo de exportación utilizado por defecto si el parámetro tipoExport se omite.*

Ejemplo

El siguiente ejemplo puede utilizarse para mostrar "Hello World" en una imagen 4D:

```
C_PICTURE(vpict)
$svg:=DOM Create XML Ref("svg";"http://www.w3.org/2000/svg")
$ref:=DOM Create XML element($svg;"text";"font-size";26;"fill";"red")
DOM SET XML ATTRIBUTE($ref;"y";"1em")
DOM SET XML ELEMENT VALUE($ref;"Hello World")
SVG EXPORT TO PICTURE($svg;vpict;Copy XML data source)
DOM CLOSE XML($svg)
```

SVG Find element ID by coordinates

SVG Find element ID by coordinates ({* ;} objetoImagen ; x ; y) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, pictureObject es un nombre de objeto (cadena) Si se omite, pictureObject es un campo o variable
objetoImagen	Imagen	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
x	Entero largo	➡	X coordenada en píxeles
y	Entero largo	➡	Y coordenada en píxeles
Resultado	Cadena	⮌	ID del elemento encontrado en la ubicación X,Y

Descripción

El comando SVG Find element ID by coordinates devuelve la identificación ("id" o atributo "xml:id") del elemento XML encontrado en la ubicación definida por las coordenadas (x,y) en la imagen SVG designada por el parámetro objetoImagen. Este comando puede utilizarse particularmente para crear interfaces gráficas interactivas utilizando objetos SVG.

Nota: para mayor información sobre el formato SVG, por favor consulte la sección **Presentación de los comandos XML utilitarios** .

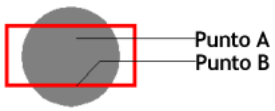
Si pasa el parámetro opcional *, indique que el parámetro pictureObject es un nombre de objeto (cadena). Si no pasa este parámetro, indique que el parámetro pictureObject es un campo o variable. En este caso, no pase una cadena sino una referencia de campo o variable (campo o variable objeto únicamente).

Note que no es obligatorio para la imagen a mostrar en un formulario. En este caso, la sintaxis de tipo "object name" no es válida y debe pasar un nombre de campo o variable.

Las coordenadas pasadas en los parámetros x y y deben estar expresadas en píxeles relativos a la esquina superior izquierda de la imagen (0,0). En el contexto de una imagen mostrada en un formulario, puede utilizar los valores devueltos por las variables sistema MouseX y MouseY. Estas variables son actualizadas en los eventos formulario On Clicked, On Double Clicked y On Mouse Up, como también en los eventos de formulario On Mouse Enter y On Mouse Move.

Nota: en el sistema de coordenadas de la imagen, [x;y] siempre especifica el mismo punto, sin importar el formato de visualización de la imagen, excepto en el caso del formato "Replicated".

El punto que se tiene en cuenta es el primer punto alcanzado. Por ejemplo, en el siguiente caso, el comando devolverá la identificación del círculo si las coordenadas del punto A se pasan y la del rectángulo si las coordenadas del punto B se pasan:



Cuando las coordenadas corresponden a un objeto superpuesto o compuesto, el comando devuelve la identificación del primero objeto con un identificador de atributo válido, si es necesario, entre los elementos padre.

El comando devuelve una cadena vacía si:

- la raíz se alcanza sin encontrar un atributo "id",
- el punto de coordenadas no pertenece a ningún objeto,
- el atributo "id" es una cadena vacía.

Nota: este comando no puede detectar objetos cuyo valor de opacida (atributo "fill-opacity") sea inferior a 0.01.

Variables y conjuntos del sistema

Si objetoImagen no contiene una imagen SVG válida, el comando devuelve una cadena vacía y la variable sistema OK toma el valor 0. De lo contrario, si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1.

SVG Find element IDs by rect

SVG Find element IDs by rect ({* ;} objetoImagen ; x ; y ; ancho ; alto ; arrIDs) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objetoImagen es un nombre de objeto (cadena). Si se omite, objetoImagen es un campo o una variable.
objetoImagen	Imagen	➡	Nombre del objeto (si se especifica *) o Campo o Variable (si se omite *)
x	Entero largo	➡	Coordenada horizontal de la esquina superior izquierda del rectángulo de selección.
y	Entero largo	➡	Coordinada vertical de la esquina superior izquierda del rectángulo de selección
ancho	Entero largo	➡	Ancho del rectángulo de selección
alto	Entero largo	➡	Altura del rectángulo de selección
arrIDs	Array texto	⬅	IDs de los elementos cuyo rectángulo circundante está en intersección con el rectángulo de selección
Resultado	Booleano	↩	True = se encontró al menos un elemento

Descripción

*El comando **SVG Find element IDs by rect** llena el array texto arrIDs con los IDs (atributo "id" o "xml:id") de los elementos XML cuyo rectángulo circundante está en intersección con el rectángulo de selección en la ubicación definida por los parámetros x y y.*

El comando devuelve True si al menos se encuentra un elemento (en otras palabras si el array arrIDs no está vacío) y de lo contrario devuelve False.

Este comando permite administrar interfaces gráficas interactivas.

*Si pasa el parámetro opcional *, indica que el parámetro objetoImagen es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objetoImagen es un campo o una variable. En este caso, pase una referencia de campo o variable (campo o variable objeto únicamente) en lugar de una cadena.*

*Si trabaja con un campo o variable imagen, el comando utiliza la imagen de origen, correspondiente a la fuente de datos. Sin embargo, si trabaja con un objeto de formulario, el comando utiliza la imagen actual, que puede modificarse vía el comando **SVG SET ATTRIBUTE** y que se conserva con las propiedades del objeto de formulario.*

Las coordenadas pasadas en los parámetros x y y deben expresarse en píxeles en relación con la esquina superior izquierda de la imagen (0,0). Puede utilizar los valores devueltos por las MouseX y MouseY. Estas variables son actualizadas en los eventos de formulario On Clicked y On Double Clicked como también en los eventos de formulario On Mouse Enter y On Mouse Move.

***Nota:** en el sistema de coordenadas de las imágenes, [x;y] siempre define el mismo punto, sin importar el formato de visualización de la imagen, excepto por el formato "Replicado".*

Todos los elementos cuyo rectángulo circundante está en intersección con el rectángulo de selección son tenidos en cuenta, incluso los que están bajo otros elementos.

SVG GET ATTRIBUTE

SVG GET ATTRIBUTE ({* ;} objetoImagen ; id_Element ; nomAtrib ; valorAtrib)

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objetoImagen es un nombre de objeto (cadena). Si se omite, objetoImagen es una variable o un campo
objetoImagen	Imagen	➡	Nombre del objeto (si se especifica *) o variable o campo (si se omite *)
id_Element	Texto	➡	ID del elemento donde uno o más atributos se definen
nomAtrib	Cadena	➡	Atributo cuyo valor quiere obtener
valorAtrib	Cadena, Entero largo	⬅	Valor actual del atributo

Descripción

*El comando **SVG GET ATTRIBUTE** se utiliza para obtener el valor actual del atributo nomAtrib en un objeto o una imagen SVG.*

*Si pasa el parámetro opcional *, indica que el parámetro objetoImagen es un nombre de objeto (cadena). En este caso, el comando devuelve el valor del atributo de la imagen renderizada adjunta al objeto. Este valor puede haber sido modificada por **SVG SET ATTRIBUTE**, por ejemplo.*
*Si no pasa el parámetro *, indica que el parámetro objetoImagen es una variable o un campo. Por lo tanto, se pasa una referencia de variable (variable objeto únicamente) o de campo en lugar de una cadena. En este caso, el comando devuelve el valor del atributo de la imagen inicial renderizada (correspondiente a la fuente de datos de la variable).*

***Nota**: este principio también se aplica al comando **SVG Find element ID by coordinates**.*

El parámetro id_Element se utiliza para definir la identificación (atributo "id" o "xml: id") del elemento cuyo valor de atributo desea obtener.

*Para obtener más información acerca de los atributos SVG, consulte la descripción del comando **SVG SET ATTRIBUTE**. Esta es una lista de atributos 4D reservados y dedicados a la animación:*

Atributos	Acceso	Comentario
4D-text	lectura/escritura	Reemplaza/lee el contenido del nodo de texto. Utilizable con los elementos 'text' 'tspan' y 'textArea'
4D-bringToFront	escritura	Si 'true', mover el nodo en frente de los nodos hermanos. Sólo se puede usar con el comando **SVG SET ATTRIBUTE**
4D-isOfClass-{IDENT [[S|COMMA] IDENT]*}	lectura	Devuelve 'true' si el atributo de clase heredado del nodo contiene todos los nombres de clases; de lo contrario, devuelve 'false'. Devuelve por ejemplo true para "4D-isOfClass-tierra" si la clase heredada del nodo es "land department01")
4D-enableD2D	lectura/escritura	Si 'false', desactiva Direct2D para el motor de renderización SVG. De hecho, los filtros SVG no son renderizados en Direct2D sino en GDI/GDIPlus. Esta opción le permite beneficiarse de los filtros SVG incluso si la base está en Direct2D. Note que esta opción sólo se tiene en cuenta cuando una imagen ya ha sido cargada en objetoImagen. Sin embargo, como esta opción se aplica d globalmente al motor, sólo debe definirse una vez por sesión (por ejemplo con un SVG simple cargado en memoria desde una variable texto al inicio de la base).

SVG SET ATTRIBUTE

SVG SET ATTRIBUTE ({* ;} objetoImagen ; id_Element ; nomAtrib ; valorAtrib {; nomAtrib2 ; valorAtrib2 ; ... ; nomAtribN ; valorAtribN} {; *})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objetoImagen es un nombre de objeto (cadena). Si se omite, objetoImagen es una variable o un campo
objetoImagen	Imagen	➡	Nombre del objeto (si se especifica *) o variable o campo (si se omite *)
id_Element	Texto	➡	ID del elemento donde uno o más atributos se definen
nomAtrib	Cadena	➡	Atributo a definir
valorAtrib	Cadena, Entero largo	➡	Nuevo valor del atributo
*	Operador	➡	Si se pasa = modificar el árbol DOM interno de la imagen SVG (variable únicamente)

Descripción

*El comando **SVG SET ATTRIBUTE** se utiliza para modificar el valor de un atributo existente en el árbol de renderización SVG de una imagen mostrada o en el árbol DOM interno de una imagen.*

*Si pasa el parámetro opcional *, indica que el parámetro objetoImagen es un nombre de objeto (cadena). En este caso, el comando aplica a los parámetros de la imagen renderizada adjunta al objeto (tenga en cuenta que los parámetros y por lo tanto la imagen renderizada del objeto sólo se crean si el comando **SVG SET ATTRIBUTE** se llama al menos una vez).*
*Si no pasa el parámetro *, indica que el parámetro objetoImagen es una variable o un campo. Por lo tanto, se pasa una referencia de variable (variable objeto únicamente) o de campo en lugar de una cadena. En este caso, el comando aplica a las imágenes renderizadas de todos los objetos que utilizan la variable o el campo.*

*Por defecto, las modificaciones realizadas por este comando aplican únicamente a las imágenes renderizadas; no se almacenan en la fuente de datos (árbol DOM interno) y se pierden cuando la imagen se borra por programación o cuando se cierra el formulario. Sin embargo, cuando transfiere esos cambios al árbol DOM interno de la imagen cuando el parámetro objetoImagen referencia a una variable: sólo necesita pasar un segundo * como parámetro final. Esto le permite conservar las modificaciones realizadas rápidamente.*

***Notas**:*

- *Transferir modificaciones en el árbol DOM interno no es posible cuando el parámetro objetoImagen referencia a un objeto.*
- *Para poder transferir las modificaciones, la variable SVG debe haber sido creada a partir de un documento DOM (con **DOM EXPORT TO VAR**). Si la variable SVG fue creada a partir de un archivo, cuando pase el segundo parámetro *, el comando no hace nada y se genera un error porque, en este caso, la fuente de datos no contiene un documento DOM modificable.*
- *Para modificar la fuente de datos de una imagen SVG, puede también utilizar los comandos **XML DOM** o el componente **4D SVG** de 4D.*

El parámetro id_Element se utiliza para definir el identificador (atributo "id" o "xml: id") del elemento cuyo(s) atributo(s) desea modificar.

En los parámetro nomAtrib y valorAtrib, pase respectivamente, el atributo a escribir y su valor (como variables, campos o valores literales). Puede pasar tantos pares de atributos/valores como quiera.

*El comando **SVG SET ATTRIBUTE** se utiliza para modificar (pero no para añadir o eliminar) la mayoría de los atributos SVG, como por ejemplo, 'fill', 'opacity', 'font-family', etc. Para una definición completa de los atributos SVG, consulte los documentos de referencia disponibles en la Internet, por ejemplo: http://www.w3.org/TR/SVG11/attindex.html. La imagen renderizada se actualiza inmediatamente, las modificaciones se transfieren a los elementos hijos para los estilos heredados.*

Tenga en cuenta que por razones técnicas, los atributos de ciertos elementos, así como ciertos atributos no pueden modificarse. La siguiente tabla muestra los elementos modificables y no modificables, así como los atributos no modificables:

Elementos cuyos atributos son modificables

svg	Restricciones: - "width" y "height" no son modificables (1) - "viewBox" sólo puede modificarse si "width" y "height" se especifican en el documento original
g	
defs	
use	
filter	Restricción: los elementos hijos fe_xxx child no son modificables
circle	
ellipse	
line	
polyline	
polygon	
path	
rect	
text, tspan, textArea	El atributo especifico "4d-text" se utiliza para modificar el texto de un elemento "text", "tspan" o "textArea" (ver el ejemplo)
Image	

Elementos cuyos atributos no pueden modificarse

linearGradient, radialGradient, Stop, solidColor, marker, symbol, clipPath, filter y los elementos que comienzan por fe, style, pattern	*Este grupo designa todos los elementos referenciables o contenidos en un elemento referenciable. Esto significa que no es posible, por ejemplo, redefinir los atributos de un gradiente (pero es posible cambiar el gradiente utilizado). Del mismo modo, para cambiar un marcador de color negro por un marcador rojo, es necesario definir ambos marcadores en el documento SVG (uno negro y uno rojo) y seleccionar uno de ellos. No es posible por ejemplo modificar el color de un rectángulo si su padre es un elemento símbolo o marcador[# / table]*
Atributos no modificables	
[#table]id o xml:id	
lang o xml:lang	
class o xml:class	
width, height	*Concierne los atributos del elemento 'svg' únicamente (1)*

(1) Estos atributos no pueden modificarse porque definen y estructuran la imagen resultante. Los atributos de width y height del elemento svg se utilizan para definir las dimensiones iniciales de la imagen en 4D y estas dimensiones debe permanecer constantes después de la creación de la imagen (sin embargo es posible modificar las dimensiones de la imagen resultante con el comando ***TRANSFORM PICTURE*** *de 4D).*

Consulte también la descripción del comando ***SVG GET ATTRIBUTE*** *para ver la lista de los atributos 4D reservados y dedicados a la animación.*

Si intenta modificar el atributo de un elemento que no es compatible o uno de sus elementos hijos, el comando no hace nada y no se genera error.

Si el comando no se ejecuta en el contexto de un formulario o si se pasa un objetoImagen inválido, la variable OK toma el valor 0. Si el comando se ha ejecutado correctamente, toma el valor 1.

Ejemplo

Modificación del contenido de un elemento de tipo texto:

```
SVG SET ATTRIBUTE(*;picture_object_name;text_element_ID;"4d-text";"Este es un texto")
```

Nota: *no hay no namespace para que el atributo pueda utilizarse en una hoja de estilo CSS sin riesgo de conflicto.*

SVG SHOW ELEMENT

SVG SHOW ELEMENT ({* ;} objetoImagen ; id {; margen})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica objetoImagen es un nombre de objeto (cadena). Si se omite, objetoImagen es una variable
objetoImagen	Imagen	➡	Nombre del objeto (si se especifica *) o variable o campo (si se omite *)
id	Texto	➡	Atributo id del elemento a visualizar
margen	Entero largo	➡	Margen de visibilidad (en píxeles por defecto)

Descripción

*El comando **SVG SHOW ELEMENT** mueve el documento SVG objetoImagen para mostrar el elemento cuyo atributo "id" es especificado por el parámetro id.*

*Si pasa el parámetro opcional *, indica que el parámetro objetoImagen es un nombre de objeto (cadena). En este caso, el comando se aplica a la imagen renderizada adjunta al objeto. Si no pasa este parámetro, indica que el parámetro objetoImagen es un campo o una variable y se pasa una referencia de variable (variable objeto únicamente) o de campo en lugar de una cadena. En este caso, el comando se aplica a las imágenes renderizadas de todos los objetos que utilizan la variable o el campo (pero no a la imagen renderizada inicial).*

El comando mueve el documento SVG para que todo el objeto, cuyos límites están definidos por su rectángulo de delimitación, sea visible. El parámetro margen se utiliza para configurar la amplitud del movimiento al especificar la distancia que debe separar el objeto que se muestra desde los bordes del documento. En otras palabras, el rectángulo aumentará en margen píxeles de ancho y de altura. Por defecto, el valor de desplazamiento es de 4 píxeles.

Este comando sólo tiene efecto en modo de visualización "top left" (con barras de desplazamiento).

Si este comando no se ejecuta en el contexto de un formulario o si se pasa un objetoImagen inválido, la variable OK toma el valor 0. Si el comando se ejecuta correctamente, toma el valor 1.

Tabla

- *Current default table*
- *Current form table*
- *DEFAULT TABLE*
- *NO DEFAULT TABLE*

Current default table

Current default table -> Resultado

Parámetro	*Tipo*	*Descripción*
Resultado	*Puntero*	*Puntero hacia la tabla por defecto*

Descripción

Current default table *devuelve un puntero hacia la tabla pasada a la última llamada de* ***DEFAULT TABLE*** *para el proceso actual.*

Ejemplo

Siempre y cuando se haya definido una tabla por defecto, la siguiente línea de código asigna como título de la ventana al nombre de la tabla por defecto actual:

```
SET WINDOW TITLE(Table name(Current default table))
```

Current form table

Current form table -> Resultado

Parámetro	Tipo		Descripción
Resultado	Puntero		Puntero hacia la tabla del formulario de salida actual

Descripción

*El comando **Current form table** devuelve el puntero hacia la tabla del formulario en pantalla o impreso en el proceso actual.*
La función devuelve Nil en los siguientes casos:

- *No hay formulario mostrado o impreso en el proceso actual,*
- *El formulario actual es un formulario de proyecto.*

Si hay varias ventanas abiertas en el proceso actual (significa que la última ventana abierta es la ventana activa actual), el comando devuelve un puntero hacia la tabla del formulario que aparece en la ventana activa.
Si el formulario mostrado actualmente es el formulario detallado de un área de subformulario, usted se encuentra en entrada de datos e hizo doble clic en un registro o subregistro en un área de subformulario de doble clic. En este caso, el comando devuelve:

- *El puntero hacia la tabla mostrada en el área del subformulario, si el subformulario muestra una tabla.*
- *Un puntero no significativo, si el área de subformulario muestra una subtabla.*

Ejemplo

En su aplicación, utiliza la siguiente convención cuando visualiza un registro:
Si la variable vsCurrentRecord está presente en un formulario, muestra "Nuevo registro" si está trabajando con un nuevo registro. Si está trabajando con el registro 56 de una selección compuesta por 5200 registros, muestra "56 de 5200".
Para esto, utilice el método de objeto para crear la variable vstRegistroActual, luego cópielo y péguelo en todos sus formularios:

```
` Método de objeto de la variable no editable vstRegistroActual
Case of
   :(Form event=On Load)
      C_STRING(31;vstRegistroActual)
      C_POINTER($vpTablaPadre)
      C_LONGINT($vlNumRegistro)
      $vpTablaPadre:=Current form table
      $vlNumRegistro:=Record number($vpTablaPadre->)
      Case of
         :($vlNumRegistro=-3)
            vstRegistroActual:="Nuevo registro"
         :($vlNumRegistro=-1)
            vstRegistroActual:="Ningún registro"
         :($vlNumRegistro>=0)
            vstRegistroActual:=String(Selected record number($vpTablaPadre->))+" de "+
            vstRegistroActual:=vstRegistroActual+String(Records in selection($vpTablaPadre->))
      End case
End case
```

DEFAULT TABLE

DEFAULT TABLE (tabla)

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla a definir como tabla por defecto

Descripción

***Tip**: aunque utilizar **DEFAULT TABLE** y omitir el nombre de la tabla pueden hacer el código más fácil de leer, muchos programadores consideran que la utilización de este comando en realidad trae más inconvenientes que ventajas. En particular, note que **DEFAULT TABLE** es prioritario cuando utiliza por ejemplo el comando **DIALOG** con un formulario proyecto y hay un formulario de la tabla por defecto con el mismo nombre.*

***DEFAULT TABLE** define tabla como la tabla por defecto para el proceso actual.*

*Un proceso no tiene tabla por defecto hasta que el comando **DEFAULT TABLE** se ejecuta. Después de que se define una tabla por defecto, cualquier comando que omita el parámetro tabla funcionará sobre la tabla por defecto. Por ejemplo, considere este comando:*

```
FORM SET INPUT([Tabla];"formulario")
```

Si [Tabla] fue definida previamente como la tabla por defecto el mismo comando podría escribirse de esta forma:

```
FORM SET INPUT("formulario")
```

*Una de las razones para definir la tabla por defecto es crear código que no sea específico para una tabla. Esto permite operar el mismo código para diferentes tablas. También puede utilizar punteros hacia tablas para escribir código que no se específico de tablas. Para mayor información sobre esta técnica, vea la descripción del comando **Table name**.*

***DEFAULT TABLE** no permite la omisión de nombres de tablas cuando se refiere a los campos. Por ejemplo:*

```
[Mi Tabla]Mi Campo:="Una Cadena" ` Correcto
```

no puede ser escrita como:

```
DEFAULT TABLE([Mi Tabla])
Mi Campo:="Una cadena" ` INCORRECTO
```

porque una tabla por defecto ha sido definida. Sin embargo, puede omitir el nombre de la tabla cuando se refiera a los campos en los triggers, en los formularios, y en los objetos que pertenecen a la tabla.

*En 4D, todas las tablas están "abiertas" y listas para ser utilizadas. **DEFAULT TABLE** no abre una tabla, define una tabla actual, o prepara la tabla para entrada o salida. **DEFAULT TABLE** es simplemente una facilidad de programación para facilitar la digitación y lectura del código.*

Ejemplo

*El siguiente ejemplo presenta primero el código sin el comando **DEFAULT TABLE**. Luego muestra el mismo código, con **DEFAULT TABLE**. El código es un bucle comúnmente utilizado para añadir nuevos registros a una base de datos. Los comandos **FORM SET INPUT** y **ADD RECORD** necesitan una tabla como primer parámetro:*

```
FORM SET INPUT([Clientes];"Añadir Registros")
Repeat
   ADD RECORD([Clientes])
Until(OK=0)
```

En este código se especifica el resultado de la tabla por defecto:

```
DEFAULT TABLE([Clientes])
FORM SET INPUT("Añadir Registros")
Repeat
   ADD RECORD
Until(OK=0)
```

NO DEFAULT TABLE

NO DEFAULT TABLE
Este comando no requiere parámetros

Descripción

*El comando **NO DEFAULT TABLE** permite cancelar el efecto del comando **DEFAULT TABLE**. Después de la ejecución de este comando, no hay tabla por defecto definida para el proceso.*

*Este comando no tendrá efecto si el comando **DEFAULT TABLE** no ha sido llamado de antemano.*

*Este comando está relacionado con el uso de formularios de proyecto (formularios no asociados a tablas): la mayoría de los comandos relacionados a los formularios (aparte de los formularios de usuario) aceptan un parámetro opcional de tipo tabla como primer parámetro. Por ejemplo, este es el caso de los comandos **FORM GET PARAMETER**, **Open form window** o **DIALOG**. Como un formulario de proyecto y un formulario de tabla pueden tener el mismo nombre, este parámetro puede utilizarse para determinar el formulario a utilizar: pase el parámetro cuando quiera apuntar a un formulario tabla y omítalo en el caso de un formulario de proyecto.*

En una base que contiene un formulario de proyecto llamado "ElFormulario" y un formulario de tabla con el mismo nombre para la tabla [Tabla1]:

```
DIALOG([Tabla1];"ElFormulario") `4D utiliza el formulario de tabla
DIALOG("ElFormulario") `4D utiliza el formulario de proyecto
```

*Sin embargo, este principio es nulo e inválido si el comando **DEFAULT TABLE** se ejecuta cuando la base contiene un formulario de proeycto y un formulario de tabla con el mismo nombre. En efecto, en este caso 4D utilizará el formulario de tabla por defecto, incluso si no se pasa el parámetro tabla. Para garantizar el uso de formularios de proyecto, simplemente utilice el comando **NO DEFAULT TABLE**.*

Ejemplo

En una base que contiene un formulario de proyecto llamado "ElFormulario" y un formulario de tabla con el mismo nombre para la tabla [Tabla1]:

```
DEFAULT TABLE([Tabla1])
DIALOG("ElFormulario") `4D utiliza el formulario de tabla
NO DEFAULT TABLE
DIALOG("ElFormulario") `4D utiliza el formulario de proyecto
```

Texto multiestilo

*El tema "**Texto multiestilo**" contiene los comandos de lenguaje utilizados para la gestión y la modificación de las áreas de texto multiestilo, también conocidas como áreas de texto enriquecido.*

*Se declara un área de texto multiestilo activando la opción "Multiestilo" en la lista de propiedades (ver **Multiestilo (área de texto enriquecido)** en el manual de Diseño). Puede utilizar el comando **OBJECT Is styled text** del tema **Objetos (Formularios)** para saber si un área de texto está en el modo multiestilo.*

Áreas 4D Write Pro

*La mayoría de los comandos del tema "**Texto con estilo**" soportan áreas 4D Write Pro. Para obtener más información, consulte la sección **Utilizar los comandos del tema Texto con estilo** en el manual 4D Write Pro Reference.*

- *Interacción de comandos genéricos con textos multiestilos*
- *Etiquetas soportadas*
- *ST COMPUTE EXPRESSIONS*
- *ST FREEZE EXPRESSIONS*
- *ST GET ATTRIBUTES*
- *ST Get content type*
- *ST Get expression*
- *ST GET OPTIONS*
- *ST Get plain text*
- *ST Get text*
- *ST GET URL*
- *ST INSERT EXPRESSION*
- *ST INSERT URL*
- *ST SET ATTRIBUTES*
- *ST SET OPTIONS*
- *ST SET PLAIN TEXT*
- *ST SET TEXT*

Interacción de comandos genéricos con textos multiestilos

Comandos de gestión de objetos texto

Los comandos que se pueden utilizar para manipular los objetos texto por programación no aceptan ningún tipo de etiquetas de estilo integradas al texto. Actúan sobre el texto mostrado como en versiones anteriores de 4D. Se trata de los siguientes comandos:

- *Tema* ***Interfaz de usuario***
  HIGHLIGHT TEXT
  GET HIGHLIGHT

Tenga en cuenta que al utilizar estos comandos con los comandos que manipulan cadenas de caracteres, es necesario filtrar los caracteres de formato utilizando el comando ***ST Get plain text****:*

```
HIGHLIGHT TEXT([Productos]Notas;1;Length(ST Get plain text([Productos]Notas))+1)
```

- *Tema* ***Objetos (Formularios)***
  Los comandos que se pueden utilizar para modificar el estilo de los objetos (por ejemplo, ***OBJECT SET FONT****) se aplican a todo el objeto y no a la selección.*
  Tenga en cuenta que si el objeto no tiene el foco cuando se ejecuta el comando, la modificación se aplica simultáneamente al objeto (el área de texto) y a su variable asociada. Si el objeto tiene el foco, la modificación se lleva a cabo en el objeto, pero no en la variable asociada. La modificación sólo se aplica a la variable cuando el objeto pierde el foco. Recuerde este principio cuando programe las áreas de texto.

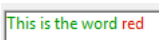
Si la opción "Guardar las etiquetas por defecto" está seleccionada para el objeto, el uso de estos comandos provocará una modificación de las etiquetas guardadas con cada objeto.

Interacción de comandos genéricos con áreas multiestilos

A partir de 4D v14, un nuevo modo de interacción se ha definido entre los comandos genéricos tales como ***OBJECT SET RGB COLORS*** *o* ***OBJECT SET FONT STYLE*** *y las áreas de texto multi-estilo.*

En las versiones anteriores de 4D, la ejecución de uno de estos comandos modificaba el contenido de las etiquetas de estilo personalizadas, insertadas en el área. A partir de ahora, sólo las propiedades por defecto se ven afectadas por estos comandos (así como las propiedades guardadas vía las etiquetas por defecto). Las etiquetas de estilo personalizadas se quedan como están.

Por ejemplo, dada un área multi-estilo, donde se guardan las etiquetas por defecto:



El texto plano del área es el siguiente:

```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#009900">This is the word <span
style="color:#D81E05">rojo</span></span>
```

Si ejecuta el siguiente código:

```
OBJECT SET COLOR(*;"myArea";-(Blue+(256*Yellow)))
```

Con 4D v14, el color rojo se mantiene:

4D v14

```
<span style="text-align:left;font-family:'Segoe
UI';font-size:9pt;color:#0000FF">This is the
word <span style="color:#D81E05">red</span>
</span>
```

versiones anteriores

```
<span style="font-family:'Segoe UI';font-size:9pt;text-align:left;font-
weight:normal;font-style:normal;text-decoration:none;color:#0000FF;">
<span style="background-color:#FFFFFF">This is the red word</span>
</span>
```

Los siguientes comandos genéricos están relacionados:

OBJECT SET RGB COLORS
OBJECT SET COLOR
OBJECT SET FONT
OBJECT SET FONT STYLE
OBJECT SET FONT SIZE

*En el contexto de las áreas de texto multiestilos, los comandos genéricos deben utilizarse sólo para definir estilos por defecto. Para administrar estilos durante la ejecución de las base, recomendamos utilizar los comandos del tema "****Texto multiestilo****".*

Comando Get edited text

Cuando se utiliza con un área de texto enriquecido, el comando ***Get edited text*** *(tema* ***Eventos de formulario****) devuelve el texto del área actual incluyendo las posibles etiquetas de estilo.*

Para recuperar el texto "plano" (texto sin etiquetas) que se está editando, debe utilizar el comando ***ST Get plain text****:*

```
ST Get plain text(Get edited text)
```

Comandos de búsqueda y ordenación

Las búsquedas y las ordenaciones efectuadas entre los objetos multiestilos tienen en cuenta las posibles etiquetas de estilo guardadas en el objeto. Si una modificación de estilo se ha hecho dentro de una palabra, la búsqueda de la palabra no tendrá éxito.

Para poder efectuar búsquedas y ordenaciones válidas, debe utilizar el comando ***ST Get plain text****. Por ejemplo:*

```
QUERY BY FORMULA([MiTabla];ST Get plain text([MiTabla]MicampoEstilo)="muy bien")
```

Normalización automática de fines de líneas

Con el fin de asegurar una mayor compatibilidad multi-plataforma de los textos manipulados en la base de datos, a partir de v14, 4D normaliza automáticamente los finales de línea de forma que ocupen un solo carácter: '\r' (retorno de carro). Esta normalización se lleva a cabo a nivel de los objetos de formulario que alojan texto multiestilo o texto bruto (variables o campos). Los finales de línea no nativos, o que utilizan una mezcla de varios caracteres (por ejemplo, '\r\n'), se consideran como una sola '\r'.

Tenga en cuenta conforme al estándar XML (formato de textos multi-estilos), los comandos de texto multiestilo también normalizan los finales de línea de las variables texto no asociadas a los objetos.

Este principio hace que sea más fácil de usar comandos de texto multiestilo o de tipo ***HIGHLIGHT TEXT*** *en un contexto multi-plataforma. Sin embargo, debe tener esto en cuenta en sus procesos cuando se trabaja con textos de fuentes heterogéneas.*

Etiquetas soportadas

Etiquetas dinámicas

Puede utilizar las siguientes etiquetas en las áreas de texto multiestilo de 4D.

Expresión 4D

* *

Esta etiqueta inserta una expresión 4D (expresión, método, campo, variable, comando, etc.) en el texto. La expresión se tokeniza y evalúa:

- *cuando se inserta la expresión*
- *cuando el objeto se carga*
- *cuando la acción estándar computeExpressions se llama desde un objeto de interfaz o por el comando **INVOKE ACTION***
- *cuando se ejecuta el comando **ST COMPUTE EXPRESSIONS***
- *cuando se ejecuta el comando**ST FREEZE EXPRESSIONS**, si se pasa el segundo parámetro *.*

El valor evaluado de la expresión no se guarda en la etiqueta , sólo su referencia.

***Nota:** para asegurar una evaluación correcta de la expresión independientemente del lenguaje o de la versión de 4D, se recomienda utilizar la sintaxis tokenizada para los elementos cuyo nombre puede variar entre las diferentes versiones (comandos, tablas, campos, constantes). Por ejemplo, para insertar el comando **Current time**, introduzca **'Current time:C178'**. Para más información, consulte **Utilizar tokens en fórmulas**.*

URL

Visible label

Esta etiqueta inserta un URL en el texto. Ejemplo:

```
<span><a href="http://www.4d.com/">4D Web Site</a></span>
```

Enlace usuario

Haga clic aquí

Los "enlaces usuario" se ven igual que las URLs, pero al hacer clic en ellos, no abren automáticamente la fuente. Puede pasar cualquier cadena que desee como referencia, y es responsabilidad del desarrollador programar las acciones personalizadas que se producen cuando se hace clic.
*Esto significa que puede crear enlaces que no son URLs, sino referencias de archivos, de métodos 4D, etc., que puede abrir o ejecutar cuando se hace clic en ellos. El comando **ST Get content type** detecta si se ha hecho clic en un enlace usuario.*

*Los enlaces usuario se definen utilizando el comando **ST SET TEXT**. Por ejemplo:*

```
ST SET TEXT(txtVar;"Este es un enlace usuario: <span style=\"-d4-ref-user:'UserLink'\">User Label</span>";$start;$end)
```

Etiquetas personalizadas

Ahora puede insertar cualquier etiqueta en el texto plano, por ejemplo <img src="http://doc.4d.com/pictures/ja.png". Se almacena en el código del texto plano sin que puedan entenderse o mostrado. Esto es particularmente útil en el contexto de mensajes de correo electrónico en formato HTML que incluyen imágenes, por ejemplo.

Etiquetas de estilo

Este párrafo lista los atributos de las etiquetas <SPAN> soportadas por 4D en las áreas de texto enriquecido. Puede utilizar estas etiquetas para implementar una gestión personalizada de estilos. Sólo las etiquetas listadas a continuación son soportadas por 4D para las variaciones de estilo.

Nombre de fuente

<SPAN STYLE="font-family: DESDEMONA"> ... </SPAN>

Tamaño de fuente

<SPAN STYLE="font-size: 20pt"> ... </SPAN>

Estilo de fuente

- ***Negrita***
  <SPAN STYLE="font-weight: bold"> ... </SPAN>
- ***Itálica o normal***
  <SPAN STYLE="font-style: italic"> ... </SPAN>
  <SPAN STYLE="font-style: normal"> ... </SPAN>
- ***Subrayado***
  <SPAN STYLE="text-decoration: underline"> ... </SPAN>
- ***Tachado***
  <SPAN STYLE="text-decoration:line-through">...</SPAN>
  ***Nota:** el estilo "tachado" no es soportado en Mac OS, pero esta etiqueta puede manejarse por programación*

Colores de fuente

<SPAN STYLE="color:green"> ... </SPAN>
o
<SPAN STYLE="color:#006CCC">...</SPAN>

Colores de fondo (Windows únicamente)

<SPAN STYLE="background-color:green"> ... </SPAN>
o
<SPAN STYLE="background-color:#006CCC">...</SPAN>

Nota: *bajo Mac OS, este atributo es ignorado. Se elimina cuando el objeto se modifica.*

Valores de colores

Para los atributos de color de fuente y de fondo, el valor de color puede ser el código hexadecimal de los colores RGB o el nombre de uno de los 16 colores HTML definidos para el estándar CSS por el W3C:

Color								
Name	Aqua	Black	Blue	Fuchsia	Gray	Green	Lime	Maroon
RGB	#00FFFF	#000000	#0000FF	#FF00FF	#808080	#008000	#00FF00	#800000
Color								
Name	Navy	Olive	Purple	Red	Silver	Teal	White	Yellow
RGB	#000080	#808000	#800080	#FF0000	#C0C0C0	#008080	#FFFFFF	#FFFF00

ST COMPUTE EXPRESSIONS

ST COMPUTE EXPRESSIONS ({* ;} objeto {; inicioSel {; finSel}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es el nombre de un objeto (cadena). Si se omite es un campo o una variable.
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección

Descripción

El comando ***ST COMPUTE EXPRESSIONS*** *actualiza las expresiones dinámicas 4D encontradas en un campo o variable ya sea 4D Write Pro o multiestilo designado por el parámetro objeto.*

Para más información acerca de las expresiones 4D utilizables en áreas de texto de multiestilo o 4D Write Pro, consulte la descripción del comando ***ST INSERT EXPRESSION****.*

El comando vuelve a evaluar el resultado de las expresiones presentes en el objeto en función del contexto actual y muestra el resultado obtenido. Por ejemplo, si la expresión introducida es la hora, el valor se modifica cada vez que se llama al comando ***ST COMPUTE EXPRESSIONS****. Las expresiones también se calculan:*

- *cuando se insertan*
- *cuando se carga el objeto*
- *cuando están "congeladas" con el comando* ***ST FREEZE EXPRESSIONS****, si se pasa el segundo parámetro *.*

ST COMPUTE EXPRESSIONS *no modifica el texto con estilo (que contiene etiquetas span), sino sólo el texto sin formato mostrado en objeto. Los valores calculados no se almacenan en el texto con estilo, sólo su referencia se almacena allí.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o de variable en lugar de una cadena.*

No es necesario que objeto tenga el foco. Sin embargo, el objeto designa un área de texto multiestilo, debe incluirse en un formulario, o de lo contrario el comando ***ST COMPUTE EXPRESSIONS*** *no tiene ningún efecto.*

Nota: *si el objeto designa un documento 4D Write Pro, será computado por el comando aunque no se abra en un objeto de formulario. Esto es necesario ya que el comando* ***ST COMPUTE EXPRESSIONS*** *puede modificar el documento convirtiendo las expresiones en expresiones de imagen después de la evaluación (si las hay), para manejarlas correctamente (ver* ***Insertar expresiones imagen****).*

Los parámetros opcionales inicioSel y finSel designan una selección de texto en objeto. Los valores inicioSel y finSel expresan una selección de texto plano, sin tener en cuenta las etiquetas de estilo o referencias que puedan estar presentes. Tenga en cuenta que una referencia es equivalente a un único carácter.

- *Si pasa inicioSel y finSel,* ***ST COMPUTE EXPRESSIONS*** *sólo actualiza las expresiones situadas dentro de esta selección.*
- *Si pasa únicamente inicioSel o si el valor de finSel es mayor que el número total de caracteres en el objeto, todas las expresiones entre inicioSel y el final del texto se calculan.*
- *Si omite inicioSel y finSel, todas las expresiones incluidas en la selección usuario de objeto se calculan.*

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "****Texto multiestilo****":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia de campo o de variable, el comando se aplica a todo el texto del objeto.

Nota*: Si inicioSel es mayor que finSel (excepto cuando finSel es 0), el comando no hace nada y la variable OK toma el valor 0.*

Ejemplo

Usted desea actualizar las referencias incluidas en la selección de texto:

```
ST COMPUTE EXPRESSIONS(*;"myText";ST Start highlight;ST End highlight)
```

ST FREEZE EXPRESSIONS

ST FREEZE EXPRESSIONS ({* ;} objeto {; inicioSel {; finSel}}{; *})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección
*	Operador	➡	Si se pasa = actualizar las expresiones antes de congelarlas

Descripción

*El comando **ST FREEZE EXPRESSIONS** "congela" el contenido de las expresiones encontradas en un campo o variable ya sea 4D Write Pro o multiestilo designado por el parámetro objeto. Esta acción convierte expresiones dinámicas en texto estático o imágenes (áreas 4D Write Pro únicamente) y elimina las referencias asociadas al objeto.*

*Para más información acerca de expresiones 4D usadas en áreas de texto multiestilo o 4D Write Pro, consulte la descripción del comando **ST INSERT EXPRESSION**.*

*El comando **ST FREEZE EXPRESSIONS** almacena el valor calculado de una expresión en un momento dado. Esta operación es necesaria particularmente antes de cada uso del objeto fuera del área estilo (exportaciones, almacenamiento en un archivo de disco, impresión, etc.) ya que solo la referencia de la expresión se mantiene en el área.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (variable o campo objeto únicamente).*
Los parámetros opcionales inicioSel y finSel designan una selección de texto en objeto. Los valores inicioSel y finSel expresan una selección de texto plano, sin tener en cuenta etiquetas de estilo que pueden estar presentes.

- *Si pasa inicioSel y finSel, **ST FREEZE EXPRESSIONS** busca la URL dentro de esta selección.*
- *Si pasa únicamente inicioSel o si el valor de finSel es mayor que el número total de caracteres en el objeto, el comando busca la URL entre inicioSel y el final del texto .*
- *Si omite inicioSel y finSel, el comando busca la URL dentro de la selección de texto actual.*

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

***Nota**: si inicioSel es mayor que finSel (excepto cuando finSel es 0) , el comando no hace nada y la variable OK toma el valor 0.*

*Por defecto, las expresiones no son recalculadas antes de ser congenladas. Si quiere que la expresión se recalcule y luego se congele, debe pasar el segundo parámetro *.*

Ejemplo

Usted quiere insertar la hora actual al inicio del texto y luego congelarla antes de guardar el registro:

```
//Insertar la hora al inicio del texto
ST INSERT EXPRESSION(*;StyledText_t;"Current time";1)
//Congelamos la expresión
ST FREEZE EXPRESSIONS(*;"StyledText_t";1)
```

ST GET ATTRIBUTES

ST GET ATTRIBUTES ({* ;} objeto ; inicioSel ; finSel ; nomAtrib ; valorAtrib {; nomAtrib2 ; valorAtrib2 ; ... ; nomAtribN ; valorAtribN})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (se se especifica *) o Variable o campo (si se omite *)
inicioSel	Entero largo	➡	Inicio de la selección de texto
finSel	Entero largo	➡	Fin de la selección de texto
nomAtrib	Entero largo	➡	Atributo a leer
valorAtrib	Variable	⬅	Valor actual del atributo

Descripción

*El comando **ST GET ATTRIBUTES** se utiliza para recuperar el valor actual de un atributo de estilo en una selección de texto del objeto de formulario designado por objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Durante la ejecución, si el objeto tiene el foco, el comando devuelve información sobre el objeto que está siendo editado, sin embargo, cuando el objeto no tiene el foco, el comando devuelve información sobre la fuente de datos (campo o variable) del objeto.*
*Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, pase una referencia campo o variable en lugar de una cadena y durante la ejecución, el comando devuelve información sobre este campo o variable.*

Los parámetros inicioSel y finSel se utilizan para designar la selección de texto de la cual leer el atributo de estilo. Pase la posición del primer carácter de la selección en inicioSel y la posición más uno del último carácter de la selección en finSel. Puede pasar 0 en finSel para designar automáticamente el último carácter del texto (pase 1 en inicioSel para designar el primer carácter del texto).
Si los valores de inicioSel y finSel son iguales o si inicioSel es mayor que finSel, se devuelve un error (excepto si finSel vale 0). Los valores inicioSel y finSel no tienen en cuenta ningún tipo de etiquetas de estilo ya presentes en el área. Se evalúan sobre la base de texto sin formato (texto del cual se han filtrado las etiquetas de estilo). 4D

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

*Pase en el parámetro nomAtrib el nombre del atributo a leer y pase en el parámetro valorAtrib una variable que debe recuperar el valor actual del atributo. Para definir el parámetro nomAtrib, debe utilizar una de las constantes del tema **Atributos de texto multiestilo**.*

Constante	Tipo	Valor	Comentario
Attribute background color	Entero largo	8	attValue=Valor hexadecimal o nombre del color HTML (Windows únicamente)
Attribute bold style	Entero largo	1	attValue=0: elimina el atributo negrita de la selección attValue=1: aplica el atributo negrita a la selección
Attribute font name	Entero largo	5	attValue=nombre de la familia de la fuente (cadena)
Attribute italic style	Entero largo	2	attValue=0: elimina el atributo itálica de la selección attValue=1: aplica el atributo itálica a la selección.
Attribute strikethrough style	Entero largo	3	attValue=0: elimina el atributo tachado de la selección attValue=1: aplica el atributo tachado a la selección
Attribute text color	Entero largo	7	attValue=valores hexadecimales o nombre de color HML
Attribute text size	Entero largo	6	attValue=número de puntos(número)
Attribute underline style	Entero largo	4	attValue=0: elimina el atributo subrayado de la selección attValue=1: aplica el atributo subrayado a la selección

Puede pasar tantos pares de atributos/valores como quiera.

Si el valor del atributo nomAtrib es el mismo para toda la selección, se devuelve en valorAtrib. Si este valor es diferente o si objeto no contiene etiquetas SPAN, se devuelven los siguientes valores:

nomAtrib	***valorAtrib si atributo heterogéneo en la selección o no etiquetas SPAN***
Attribute background color	*FFFFFFFF*
Attribute bold style	*2*
Attribute font name	*"" (empty string)*
Attribute italic style	*2*
Attribute strikethrough style	*2*
Attribute text color	*FFFFFFFF*
Attribute text size	*-1*
Attribute underline style	*2*

Ejemplo

Dado un campo [Table_1]StyledText mostrado en un formulario. El objeto tiene la propiedad Multiestilo y se llama "StyledText_t". Usted quiere obtener el texto resaltado así como también el estado de atributo de estilo Negrita. Puede proceder de dos formas diferentes dependiendo de si utiliza el nombre del objeto o la referencia del campo.

- *Utilizando el nombre del objeto:*

```
$text:=ST Get text(*;"StyledText_t";ST Start highlight;ST End highlight)
ST GET ATTRIBUTES(*;"StyledText_t";ST Start highlight;ST End highlight;Attribute bold style;$bold)
```

- *Utilizando el nombre del campo:*

```
GET HIGHLIGHT([Table_1]StyledText;$Begin_l;$End_l)
$text:=ST Get text([Table_1]StyledText;$Begin_l;$End_l)
ST GET ATTRIBUTES([Table_1]StyledText;$Begin_l;$End_l;Attribute bold style;$bold)
```

Variables y conjuntos del sistema

Después de ejecutar este comando, la variable OK toma el valor 1 si no se presenta ningún error; de lo contrario, toma el valor 0. Este es el caso particularmente cuando las etiquetas de estilo no se evalúan correctamente (etiquetas incorrectas o faltantes).

En caso de error, no cambia la variable. Cuando ocurre un error en una variable cuando se está evaluando el texto, 4D transforma el texto en texto plano; como resultado, los caracteres <, > y & se convierten en entidades HTML.

ST Get content type

ST Get content type ({* ;} object {; inicioSel {; finSel {; inicioBloq {; finBloq}}}}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
object	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Campo o variable (si se omite *)
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección
inicioBloq	Entero largo	⬅	Inicio de posición del primer tipo de la selección
finBloq	Entero largo	⬅	Fin de posición del primer tipo de la selección
Resultado	Entero largo	↩	Tipo de contenido

Descripción

*El comando **ST Get content type** devuelve el tipo de contenido encontrado en el campo o la variable de texto multiestilo designada por el parámetro objeto .*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Durante la ejecución, si el objeto tiene el foco, el comando devuelve la información del objeto que se está editando, y si el objeto no tiene el foco, el comando devuelve la información de la fuente de datos del objeto (variable o campo).*
*Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena. Durante la ejecución, el comando devuelve la información de la variable o del campo.*

Los parámetros opcionales inicioSel y finSel designan una selección de texto en objeto. Los valores inicioSel y finSel expresan una selección de texto plano, sin tener en cuenta etiquetas de estilo que pueden estar presentes.

- *Si pasa inicioSel y finSel, **ST Get content type** evalúa el contenido ubicado al interior de esta selección.*
- *Si pasa únicamente inicioSel o si el valor de finSel es mayor que el número total de caracteres en el objeto, el contenido ubicado entre inicioSel y el final del texto.*
- *Si omite inicioSel y finSel, el contenido dentro ubicado al interior de la selección actual de texto se evalúa.*

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

***Nota**: si inicioSel es mayor que finSel (excepto cuando finSel es 0) , el comando no hace nada y la variable OK toma el valor 0. Los parámetros opcionales inicioBloq y finBloq recuperan la posición del primer y último carácter del primer bloque homogéneo identificado en el objeto o la selección del objeto. Por ejemplo, si la selección contiene una expresión y, a continuación texto plano, inicioBloq y finBloq devolverán los límites de la expresión. Usted puede hacer un bucle para procesar todos los bloques de la selección.*

*El comando devuelve un valor que designa el tipo de contenido identificado. Puede comparar este valor con las siguientes constantes, que se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST Expression type	Entero largo	2	La selección contiene sólo una referencia de expresión
ST Mixed type	Entero largo	3	La selección contiene al menos dos tipos de contenidos diferentes
ST Picture type	Entero largo	6	La selección contiene sólo una imagen (áreas 4D Write Pro únicamente)
ST Plain type	Entero largo	0	La selección contiene texto y ninguna referencia
ST Unknown tag type	Entero largo	4	La selección contiene sólo una etiqueta de tipo desconocido
ST URL type	Entero largo	1	La selección contiene sólo una referencia de URL
ST User type	Entero largo	5	La selección contiene sólo una referencia personalizada

Ejemplo

Usted desea mostrar los comandos de un menú contextual basado en el tipo de contenido seleccionado en el área.

```
Case of
   :(Form event=On Clicked)
 //we retrieve the selection
      GET HIGHLIGHT(*;"myText";startSel;endSel)
      If(Contextual click & (Macintosh control down=False)) //llama el menú contextual
         If(startSel=endSel) // sin contenido seleccionado
 //activamos únicamente ciertos comandos
            DISABLE MENU ITEM(<>menu_STYLEDTEXT;2)
            DISABLE MENU ITEM(<>menu_STYLEDTEXT;4)
```

```
            ENABLE MENU ITEM(<>menu_STYLEDTEXT;6)
            ...
        Else  // obtenemos el tipo de contenido
            CT_Texttype:=ST Get content type(*;"myText";startSel;endSel)
            Case of  // procesamiento de tipos diferentes
                :(CT_Texttype=ST URL type)
                    DISABLE MENU ITEM(<>menu_STYLEDTEXT;6)
                    ENABLE MENU ITEM(<>menu_STYLEDTEXT;7)
                    ...
                :(CT_Texttype=ST Expression type)
                    DISABLE MENU ITEM(<>menu_STYLEDTEXT;6)
                    DISABLE MENU ITEM(<>menu_STYLEDTEXT;7)
                    ...
                Else
                    ENABLE MENU ITEM(<>menu_STYLEDTEXT;6)
                    DISABLE MENU ITEM(<>menu_STYLEDTEXT;7)
                    ...
            End case
        End if
        GET MOUSE($xCoord;$yCoord;$ButtonState)
        $AlphaVar:=Dynamic pop up menu(<>menu_STYLEDTEXT;"";$xCoord;$yCoord)
        startSel:=-3
        endSel:=-3
    End if
    ...
End if
```

ST Get expression

ST Get expression ({* ;} objeto {; inicioSel {; finSel}}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Campo o variable (si se omite *)
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección
Resultado	Texto	⮌	Etiqueta de la expresión

Descripción

*El comando **ST Get expression** devuelve la primera expresión que se encuentra en la selección actual del campo o de la variable de texto con estilo designada por el parámetro objeto.*

El comando devuelve la etiqueta de la expresión, como fue insertada en el objeto (por ejemplo, "mymethod" o "[tabla1]campo1"). No se devuelve el valor actual de la expresión.

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Durante la ejecución, si el objeto tiene el foco, el comando devuelve la información del objeto que se está editando, y si el objeto no tiene el foco, el comando devuelve la información de la fuente de datos del objeto (variable o campo).*
*Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena. Durante la ejecución, el comando devuelve la información de la variable o del campo.*

Los parámetros opcionales inicioSel y finSel designan una selección de texto en objeto. Los valores inicioSel y finSel expresan una selección de texto plano, sin tener en cuenta etiquetas de estilo que pueden estar presentes en el texto.

- *Si pasa inicioSel y finSel, **ST Get expression** busca la expresión al interior de esta selección.*
- *Si pasa únicamente inicioSel o si el valor de finSel es mayor que el número total de caracteres en el objeto, el comando busca la expresión entre inicioSel y el final del texto .*
- *Si omite inicioSel y finSel, el comando busca la expresión dentro de la selección de texto actual.*

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

***Nota**: si inicioSel es mayor que finSel (excepto cuando finSel es 0) , el comando no hace nada y la variable OK toma el valor 0. Si no se encuentra una expresión en la selección, el comando devuelve una cadena vacía.*

Ejemplo 1

Cuando hay un evento doble clic, usted comprueba que existe una expresión, y si es así, se muestra un diálogo en el que han recuperado sus valores para que el usuario puede modificarlos:

```
Case of
   :(Form event=On Double Clicked)
      GET HIGHLIGHT(*;"StyledText_t";startSel;endSel)
      If(ST Get content type(*;"StyledText_t";startSel;endSel)=ST Expression type)
         vExpression:=ST Get expression(*;"StyledText_t";startSel;endSel)
         $winRef:=Open form window("Dial_InsertExpr";Movable form dialog box;Horizontally centered;Vertically centered;*)
         DIALOG("Dial_InsertExpr")
         If(OK=1)
            ST INSERT EXPRESSION(*;"StyledText_t";vExpression;startSel;endSel)
            HIGHLIGHT TEXT(*;"StyledText_t";startSel;endSel)
         End if
      End if
End case
```

Ejemplo 2

Usted quiere ejecutar un método 4D cuando se hace clic en un enlace de usuario:

```
Case of
   :(Form event=On Clicked)
```

```
    //recuperamos la selección
        HIGHLIGHT TEXT(*;"myText";startSel;endSel)
        If(startSel#endSel)  //hay contenido seleccionado
    //obtenemos el tipo del contenido
            $CT_type:=ST Get content type(*;"myText";startSel;endSel)
            If($CT_type=ST User type)  //this is a user link
                MyMethod  //ejecutamos un método 4D
            End if
        End if
End case
```

ST GET OPTIONS

ST GET OPTIONS ({* ;} objeto ; opcion ; valor {; opcion2 ; valor2 ; ... ; opcionN ; valorN})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Campo o variable (si se omite *)
opcion	Entero largo	➡	Opción a leer
valor	Entero largo	⬅	Valor actual de la opción

Descripción

*El comando **ST GET OPTIONS** obtiene el valor actual de una o varias opciones de funcionamiento del campo o de la variable de texto con estilo designada por el parámetro objeto.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Durante la ejecución, si el objeto tiene el foco, el comando devuelve la información del objeto que se está editando, y si el objeto no tiene el foco, el comando devuelve la información de la fuente de datos del objeto (variable o campo).*
*Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena. Durante la ejecución, el comando devuelve la información de la variable o del campo.*

*Pase el código de la opción a leer en el parámetro opcion. El comando devuelve en valor el actual de la opción. Para ambos parámetros, puede utilizar las siguientes constantes, que se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST Expressions display mode	Entero largo	1	El parámetro valor puede contener ST Values o ST References
ST References	Entero largo	1	Muestra las cadenas fuente de las expresiones
ST Values	Entero largo	0	Muestra los valores calculados de las expresiones

ST Get plain text

ST Get plain text ({* ;} objeto {; refMode}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo.
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o variable o campo (si se omite *)
refMode	Entero largo	➡	Modo para el manejo de las referencias presentes en el texto
Resultado	Texto	⬅	Texto sin etiquetas

Descripción

*El comando **ST Get plain text**remueve las etiquetas de estilo de la variable o campo de texto designado por los parámetros * y objeto y devuelve el texto plano.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Durante la ejecución, si el objeto tiene el foco, el comando devuelve información sobre el objeto que está siendo editado, sin embargo, cuando el objeto no tiene el foco, el comando devuelve información sobre la fuente de datos (campo o variable) del objeto.*
*Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, pase una referencia campo o variable en lugar de una cadena y durante la ejecución, el comando devuelve información sobre este campo o variable.*

*El parámetro opcional modoRef indica la forma en que las referencias que se encuentran en objeto deben ser devueltas. En modoRef, pase una de las siguientes constantes, que se encuentran en el tema "**Texto multiestilo**" (puede pasar una sola constante o una combinación):*

Constante	Tipo	Valor	Comentario
ST 4D Expressions as sources	Entero largo	2	Se devuelve la cadena original de las referencias de expresiones 4D
ST 4D Expressions as values	Entero largo	1	Las referencias de expresiones 4D se devuelven en su forma evaluada (funcionamiento por defecto en los formularios)
ST References as spaces	Entero largo	0	Cada referencia se devuelve como un carácter espacio sin separación (funcionamiento por defecto, utilizado por los otros comandos)
ST Tags as plain text	Entero largo	64	El rótulo de la etiqueta se devuelve en texto plano. Por ejemplo para el tag '<img src="test.jpg" alt="picture">my picture</img>', el texto plano es "my picture" (funcionamiento por defecto en los formularios)
ST Tags as XML code	Entero largo	128	El código XML de la etiqueta se devuelve en texto plano. Por ejemplo para el tag '<img src="test.jpg" alt="picture">my picture</img>', el texto plano es '<img src="test.jpg" alt="picture">my picture</img>'
ST Text displayed with 4D Expression sources	Entero largo	86	Devuelve el texto tal y como se muestra en los formularios con la cadena de origen de las expresiones 4D. Corresponde a la combinación predefinida de las constantes 2+4+16+64.
ST Text displayed with 4D Expression values	Entero largo	85	Devuelve el texto tal y como se muestra en los formularios con las expresiones 4D en su forma evaluada. Corresponde a la combinación predefinida de las constantes 1+4+16+64.
ST URL as labels	Entero largo	4	La etiqueta visible de los URLs se devuelve, por ejemplo "Visite nuestro sitio web" (funcionamiento por defecto en los formularios)
ST URL as links	Entero largo	8	Se devuelve el enlace, por ejemplo "http://www.4d.com"
ST User links as labels	Entero largo	16	Se devuelve la etiqueta visible del enlace usuario (funcionamiento por defecto en los formularios)
ST User links as links	Entero largo	32	Se devuelve el contenido del enlace usuario

***Notas**:*

- *dado que el texto plano sigue siendo el mismo, independientemente de los valores pasados en el parámetro modoRef, el parámetro modoRef opcional sólo es útil cuando el texto contiene referencias.*
- *si un documento 4D Write Pro contiene tablas, el contenido de cada celda se trata como párrafos individuales y se devuelve como texto separado por pestañas. Las filas están separadas por retornos de carro.*

Ejemplo 1

*Usted está buscando el texto "muy bien" entre los valores de un campo de texto multiestilo. El valor se guardó de la siguiente forma: "El clima está muy bien **hoy**".*

```
QUERY BY FORMULA([Comments];ST Get plain text([Comments]Weather)="@muy bien@")
```

Nota: *en este contexto, la siguiente instrucción no dará el resultado deseado porque el texto está guardad con etiquetas de estilo:*

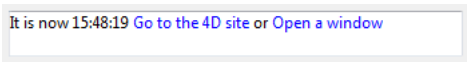
```
QUERY([Comments];[Comments]Weather="@muybien@")
```

Ejemplo 2

Dado el siguiente texto ubicado en el área multiestilo "MyArea":

```
<span>It is now <span style="-d4-ref:'Current time:C178'"> </span> <a href="http://www.4d.com">Go to the 4D site</a> or <span style="-d4-ref-user:'openW'">Open a window</span></span>
```

Este texto se muestra:



Si ejecuta el siguiente código:

```
$txt :=ST Get plain text(*;"myArea";ST References as spaces)
 // $txt = "ahora      o " (espacios)
$txt :=ST Get plain text(*;"myArea";ST 4D Expressions as values)
 // $txt = "actualmente 15:48:19   o   "
$txt :=ST Get plain text(*;"myArea";ST 4D Expressions as sources)
 // $txt = "es ahora la hora actual o   "
$txt :=ST Get plain text(*;"myArea";ST URL as links)
 //$txt = "es ahora http://www.4d.com or   "
$txt :=ST Get plain text(*;"myArea";ST Text displayed with 4D Expression values)
 //$txt = "es ahora 15:48:19 ir al sitio de 4D o abrir una ventana"
$txt :=ST Get plain text(*;"myArea";ST Text displayed with 4D Expression sources)
 //$txt = "es actualmente Hora actual ir al sitio de 4D o abrir una ventana"
$txt :=ST Get plain text(*;"myArea";ST User links as labels)
 //$txt = "es ahora      o Abrir una ventana"
$txt :=ST Get plain text(*;"myArea";ST User links as links)
 //$txt = "es ahora      u openW"
```

Variables y conjuntos del sistema

Después de ejecutar este comando, la variable OK toma el valor 1 si no se presenta ningún error; de lo contrario, toma el valor 0. Este es el caso particularmente cuando las etiquetas de estilo no se evalúan correctamente (etiquetas incorrectas o faltantes).

En caso de error, no cambia la variable. Cuando ocurre un error en una variable cuando se está evaluando el texto, 4D transforma el texto en texto plano; como resultado, los caracteres <, > y & se convierten en entidades HTML.

ST Get text

ST Get text ({* ;} objeto {; inicioSel {; finSel}}) -> Resultado

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o campo o variable texto (si se omite *)
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección
Resultado	Texto	↩	Texto incluyendo las etiquetas de estilo

Descripción

*El comando **ST Get text** devuelve el texto con estilo encontrado en el campo o variable de texto designado por el parámetro objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (cadena). Durante la ejecución, si el objeto tiene el foco, el comando devuelve información sobre el objeto que está siendo editado, sin embargo, cuando el objeto no tiene el foco, el comando devuelve información sobre la fuente de datos (campo o variable) del objeto.*
*Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, pase una referencia campo o variable en lugar de una cadena y durante la ejecución, el comando devuelve información sobre este campo o variable.*

El comando devuelve el texto con las etiquetas de estilo asociadas a él, lo que significa, por ejemplo, que puede copiar y pegar texto mientras conserva su estilo.
Los parámetros opcionales inicioSel y finSel permiten designar una selección de texto en objeto. Los valores de inicioSel y finSel ofrecen una selección de texto sin formato, sin tener en cuenta ningún tipo de etiquetas de estilo en el texto.

- *Si se omite inicioSel y endSel, **ST Get text** devuelve todo el texto en el objeto,*
- *Si pasa inicioSel y finSel, **ST Get text** devuelve la selección de texto definida por estos límites.*

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de la selección en los parámetros inicioSel y endSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

Si los valores de inicioSel y finSel son iguales o si inicioSel es mayor que finSel, se devuelve un error.

Variables y conjuntos del sistema

Después de ejecutar este comando, la variable OK toma el valor 1 si no se presenta ningún error; de lo contrario, toma el valor 0. Este es el caso particularmente cuando las etiquetas de estilo no se evalúan correctamente (etiquetas incorrectas o faltantes).
En caso de error, no cambia la variable. Cuando ocurre un error en una variable cuando se está evaluando el texto, 4D transforma el texto en texto plano; como resultado, los caracteres <, > y & se convierten en entidades HTML.

ST GET URL

ST GET URL ({* ;} objeto ; textoURL ; direccionURL {; inicioSel {; finSel}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Campo o variable (si se omite *)
textoURL	Texto	⬅	Texto visible de la URL
direccionURL	Texto	⬅	Dirección de la URL
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección

Descripción

*El comando **ST GET URL** devuelve la etiqueta y la dirección de la primera URL detectada en el campo o la variable de texto multiestilo designado por el parámetro objeto.*

La etiqueta y la dirección se devuelven en los parámetros textoURL y direccionURL. Si la selección no contiene una URL, se devuelven cadenas vacías en estos parámetros.

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Durante la ejecución, si el objeto tiene el foco, el comando devuelve la información del objeto que se está editando, y si el objeto no tiene el foco, el comando devuelve la información de la fuente de datos del objeto (variable o campo).*
*Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena. Durante la ejecución, el comando devuelve la información de la variable o del campo.*
Los parámetros opcionales inicioSel y finSel designan una selección de texto en objeto. Los valores inicioSel y finSel expresan una selección de texto plano, sin tener en cuenta etiquetas de estilo que pueden estar presentes.

- *Si pasa inicioSel y finSel, **ST GET URL** busca la URL dentro de esta selección.*
- *Si pasa únicamente inicioSel o si el valor de finSel es mayor que el número total de caracteres en el objeto, el comando busca la URL entre inicioSel y el final del texto .*
- *Si omite inicioSel y finSel, el comando busca la URL dentro de la selección de texto actual.*

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

***Nota**: si inicioSel es mayor que finSel (excepto cuando finSel es 0) , el comando no hace nada y la variable OK toma el valor 0.*

Ejemplo

Cuando hay un evento de doble clic, se comprueba que no existe en realidad una URL, y si es así, se muestra un diálogo en el que ha recuperado sus valores para que el usuario puede modificarlos:

```
Case of
   :(Form event=On Double Clicked)
      GET HIGHLIGHT(*;"StyledText_t";startSel;endSel)
      If(ST Get content type(*;"StyledText_t";startSel;endSel)=ST URL type) //URL
         ST GET URL(*;"StyledText_t";vTitle;vURL;startSel;endSel)
         $winRef:=Open form window("Dial_InsertURL";Movable form dialog box;Horizontally centered;Vertically centered;*)
         SET WINDOW TITLE("URL settings")
         DIALOG("Dial_InsertURL")
         If(OK=1)
            ST INSERT URL(*;"StyledText_t";vTitle;vURL;startSel;endSel)
            HIGHLIGHT TEXT(*;"StyledText_t";startSel;startSel+1)
         End if
      End if
End case
```

ST INSERT EXPRESSION

ST INSERT EXPRESSION ({* ;} objeto ; expresion {; inicioSel {; finSel}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o variable
objeto	Objeto	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
expresion	Texto	➡	Expresión y (opcional) formato a insertar
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección

Descripción

*El comando **ST INSERT EXPRESSION** inserta una referencia a la expresión en el campo o la variable de texto multiestilo designada por el parámetro objeto.*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (variable o campo objeto únicamente).*

En el parámetro expresión, se pasa la expresión 4D a evaluar en el objeto. Una expresión válida 4D es una cadena que retorna un valor. La expresión puede ser una campo, una variable, un comando 4D, una instrucción que retorne un valor, un método de proyecto, etc.

La expresión debe ser puesta entre comillas dobles ("").

***Nota:** insertar expresiones imagen (por ejemplo variables de tipo imagen) se soporta en áreas 4D Write Pro (ver **Insertar expresiones imagen**) pero no se soporta en áreas de texto multi estilo.*

Si expresión retorna un valor que contiene retornos de carro y tabulaciones, 4D formatea el texto de acuerdo al objeto que alberga la expresión; caracteres de retorno de carro se interpretan como rupturas de línea.

Puede dar formato a la expresión mediante la inclusión de información de formato en el parámetro de expresión. En este caso, el parámetro debe ser en la forma:

```
"String(valor;formato)"
```

... donde valor contiene a la expresión misma y formato contiene el formato a aplicar. El parámetro formato puede tener los siguientes valores:

- *para los números: por ejemplo "###,##" cualquier formato de visualización del número (existente o no).*
- *para fechas: un número que designa un formato de fecha existente. Puede utilizar las constantes del tema "**Formatos de salida de fechas**" , por ejemplo, System date short .*
- *para los tiempos: un número que designa un formato de hora existente. Puede utilizar las constantes del tema "**Formatos de salida de hora**" , por ejemplo, System time short .*

Por ejemplo:

```
"String ([tabla_1]Campo_1;System date short)"
```

*Por defecto, los **valores** de expresión se muestran en las áreas de texto multiestilo. Puede forzar la visualización de las **referencias** utilizando el comando **ST SET OPTIONS**.*

Los parámetros opcionales inicioSel y finSel designan una selección de texto en objeto. Los valores inicioSel y finSel expresan una selección de texto plano, sin tener en cuenta etiquetas de estilo que pueden estar presentes.

- *Si pasa únicamente inicioSel, el resultado de la expresión se inserta en la ubicación especificada.*
- *Si omite inicioSel y finSel, el resultado de la expresión se inserta en la ubicación del cursor.*
- *Si pasa inicioSel y finSel, **ST INSERT EXPRESSION** remplaza el texto en esta selección con el resultado de la expresion. Si el valor de finSel es mayor que el número total de caracteres en el objeto, todos los caracteres entre inicioSel y el final del texto son remplazados por el resultado de la expresion.*

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

***Nota**: si inicioSel es mayor que finSel (excepto cuando finSel es 0) , el comando no hace nada y la variable OK toma el valor 0.*

Ejemplo

Desea reemplazar el texto seleccionado con el resultado de un método proyecto:

```
ST INSERT EXPRESSION(*;"miTexto";"miMetodo";ST Start highlight;ST End highlight)
```

ST INSERT URL

ST INSERT URL ({* ;} objeto ; textoURL ; direccionURL {; inicioSel {; finSel}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o variable
objeto	Objeto de formulario	➡	Nombre de objeto (si se especifica *) o Campo o variable (si se omite *)
textoURL	Texto	➡	Texto visible del URL
direccionURL	Texto	➡	Dirección de la URL
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección

Descripción

*El comando **ST INSERT URL** inserta un enlace URL en el campo o la variable de texto con estilo designada por el parámetro objeto .*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si omite el parámetro *, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable (variable o campo objeto únicamente).*

En el parámetro textoURL, pase el texto visible de la URL, como debe aparecer en el objeto. Por ejemplo, se pueden usar etiquetas de texto como "Sitio web 4D" o "Siga este enlace para obtener más información." También puede utilizar la dirección misma, por ejemplo "http://www.4d.com".

En el parámetro direccionURL, pase la dirección completa a la cual desea conectar la página del navegador, por ejemplo "http://www.4D.com".
Los parámetros opcionales inicioSel y finSel designan una selección de texto en objeto. Los valores inicioSel y finSel expresan una selección de texto plano, sin tener en cuenta etiquetas de estilo que pueden estar presentes en el texto.

- *Si pasa únicamente inicioSel, textoURL inserta en la ubicación especificada.*
- *Si omite inicioSel y finSel, textoURL se inserta en la ubicación del cursor.*
- *Si pasa inicioSel y finSel, **ST INSERT URL** remplaza el texto en esta selección por textoURL. Si el valor de finSel es mayor que el número total de caracteres en el objeto, todos los caracteres entre inicioSel y el final del texto son remplazados por textoURL.*

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

***Nota**: si inicioSel es mayor que finSel (excepto cuando finSel es 0) , el comando no hace nada y la variable OK toma el valor 0.*
*Una vez insertado el enlace, es activa: el uso de **Ctrl+clic** (Windows) o **Comando+clic** (OS X) abre una página del navegador por defecto en la dirección especificada en el parámetro direccionURL.*

Ejemplo

Usted desea insertar un enlace al sitio web de 4D para reemplazar el texto seleccionado en el objeto:

```
vTitle:="4D Web Site"
vURL:="http://www.4d.com/"
ST INSERT URL(*;"myText";vTitle;vURL;ST Start highlight;ST End highlight)
```

ST SET ATTRIBUTES

ST SET ATTRIBUTES ({* ;} objeto ; inicioSel ; finSel ; nomAtrib ; valorAtrib {; nomAtrib2 ; valorAtrib2 ; ... ; nomAtribN ; valorAtribN})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *), o Variable o campo Texto (si se omite *)
inicioSel	Entero largo	➡	Inicio de la nueva selección de texto
finSel	Entero largo	➡	Fin de la nueva selección de texto
nomAtrib	Cadena	➡	Atributo a definir
valorAtrib	Cadena, Entero largo	➡	Nuevo valor del atributo

Descripción

*El comando **ST SET ATTRIBUTES** permite modificar uno o más atributos de estilo en el(los) objeto(s) de formulario designado(s) por objeto.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Durante la ejecución, si el objeto tiene el foco, el comando sólo se aplica al objeto que se está editando y no a su fuente de datos (campo o variable). Los cambios sólo se transfieren a la fuente (y a cualquier otro objeto con esta misma fuente) cuando el objeto que se está editando se valida, ya sea por perder el foco o con la tecla **Intro**. Si el objeto no tiene el foco, el comando se aplica directamente a la fuente de datos y los cambios son inmediatamente trasladados a otros objetos con la misma fuente.*
*Si se omite el parámetro *, indica que el parámetro objeto es un campo o una variable y se pasa una referencia de campo o variable en lugar de una cadena. El comando se aplica directamente al campo o a la variable y los cambios son transferidos inmediatamente a todos los objetos que utilizan esta fuente, incluyendo el objeto con el foco.*

***Nota**: sólo puede utilizar los atributos de estilo con campos de tipo texto. Dado que los campos de tipo Alfa tienen una longitud predefinida, la adición de etiquetas de estilo podría producir una pérdida de datos.*

*La definición de un atributo se efectúa por medio de la inserción o modificación de etiquetas HTML de estilo al interior del texto (para más información sobre este punto, consulte el Manual de Diseño). Tenga en cuenta que **ST SET ATTRIBUTES** inserta etiquetas de estilo en todos los casos, aunque objeto designe los objetos texto del formulario que no tengan la propiedad Multiestilo.*

Los parámetros inicioSel y finSel se pueden utilizar para designar a la selección de texto a la cual aplicar la modificación de estilo al interior del objeto. En inicioSel pase la posición del primer carácter a modificar y en finSel, pase la posición del último carácter a modificar más uno. Puede pasar 0 en finSel para designar automáticamente el último carácter del texto (pase 1 en inicioSel para designar el primer carácter).
Si el valor de finSel es superior al número de caracteres del objeto, todos los caracteres entre inicioSel y el final del texto se modifican. Si el valor de inicioSel es mayor que el de finSel (excepto cuando finSel vale 0), el comando no hace nada y la variable OK toma el valor 0.
Los valores de inicioSel y finSel no tienen en cuenta las etiquetas de estilo presentes en el área. Son evaluados sobre la base de texto sin formato (texto donde las etiquetas de estilo han sido filtradas).

*4D ofrece constantes predefinidas para que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

*Pase en los parámetros nomAtrib y valorAtrib respectivamente el nombre y el valor del atributo a modificar. Puede pasar tantos pares de atributos/valores como quiera. Para definir el parámetro nomAtrib, utilice las constantes predefinidas del tema **Atributos de texto multiestilo**. El valor a pasar en el parámetro valorAtrib depende del parámetro nomAtrib:*

Constante	Tipo	Valor	Comentario
Attribute background color	Entero largo	8	attValue=Valor hexadecimal o nombre del color HTML (Windows únicamente)
Attribute bold style	Entero largo	1	attValue=0: elimina el atributo negrita de la selección attValue=1: aplica el atributo negrita a la selección
Attribute font name	Entero largo	5	attValue=nombre de la familia de la fuente (cadena)
Attribute italic style	Entero largo	2	attValue=0: elimina el atributo itálica de la selección attValue=1: aplica el atributo itálica a la selección.
Attribute strikethrough style	Entero largo	3	attValue=0: elimina el atributo tachado de la selección attValue=1: aplica el atributo tachado a la selección
Attribute text color	Entero largo	7	attValue=valores hexadecimales o nombre de color HML
Attribute text size	Entero largo	6	attValue=número de puntos(número)
Attribute underline style	Entero largo	4	attValue=0: elimina el atributo subrayado de la selección attValue=1: aplica el atributo subrayado a la selección

Colores
Si pasa las constantes Attribute text color o Attribute background en nomAtrib, debe pasar en valorAtrib una cadena que contenga un nombre de color HTML o un valor de color hexadecimal:

Nombre de color HTML	Valor hexadecimal
Aqua	#00FFFF
Black	#000000
Blue	#0000FF
Fushia	#FF00FF
Gray	#808080
Green	#008000
Lime	#00FF00
Maroon	#800000
Navy	#000080
Olive	#808000
Purple	#800080
Red	#FF0000
Silver	#C0C0C0
Teal	#008080
White	#FFFFFF
Yellow	#FFFF00

Ejemplo

En este ejemplo, modificamos el tamaño y el color de texto como también los atributos negrita y subrayado de los caracteres 2 a 4 del campo:

```
ST SET ATTRIBUTES([MyTable]MyField;2;5;Attribute font name;"Arial";Attribute text size;10;Attribute underline style;1;Attribute bold style;1;Attribute text color;"Blue")
```

Variables y conjuntos del sistema

Después de ejecutar este comando, la variable OK toma el valor 1 si no se presenta ningún error; de lo contrario, toma el valor 0. Este es el caso particularmente cuando las etiquetas de estilo no se evalúan correctamente (etiquetas incorrectas o faltantes).

En caso de error, no cambia la variable. Cuando ocurre un error en una variable cuando se está evaluando el texto, 4D transforma el texto en texto plano; como resultado, los caracteres <, > y & se convierten en entidades HTML.

ST SET OPTIONS

ST SET OPTIONS ({* ;} objeto ; opcion ; valor {; opcion2 ; valor2 ; ... ; opcionN ; valorN})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es un campo o una variable
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Campo o variable (si se omite *)
opcion	Entero largo	➡	Opción a definir
valor	Entero largo	➡	Nuevo valor de la opción

Descripción

*El comando **ST SET OPTIONS** modifica una o varias opciones de funcionamiento del campo o de la variable de texto con estilo designada por el parámetro objeto .*

*Si pasa el parámetro opcional * indica que el parámetro objeto es un nombre de objeto (cadena). Si no pasa este parámetro, indica que el parámetro objeto es un campo o una variable. En este caso, se pasa una referencia de campo o variable en lugar de una cadena (campo o variable objeto únicamente).*

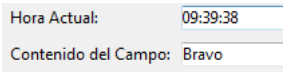
Pase el código de la opción a modificar en opcion y su nuevo valor en valor.
*El parámetro opcion soporta la siguiente constante del tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST Expressions display mode	Entero largo	1	El parámetro valor puede contener ST Values o ST References

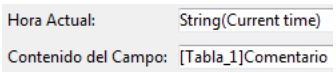
En el parámetro valor, puede pasar una de las siguientes constantes:

Constante	Tipo	Valor	Comentario
ST References	Entero largo	1	Muestra las cadenas fuente de las expresiones
ST Values	Entero largo	0	Muestra los valores calculados de las expresiones

Visualización de los valores:



Visualización de las expresiones:



Ejemplo

El siguiente código permite cambiar el modo de visualización del área:

```
ST GET OPTIONS(*;"StyledText_t";ST Expressions display mode;$exprValue)
If($exprValue=1)
   ST SET OPTIONS(*;"StyledText_t";ST Expressions display mode;ST Values)
Else
   ST SET OPTIONS(*;"StyledText_t";ST Expressions display mode;ST References)
End if
```

ST SET PLAIN TEXT

ST SET PLAIN TEXT ({* ;} objeto ; nuevTexto {; inicioSel {; finSel}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena). Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable o campo (si se omite *)
nuevTexto	Texto	➡	Texto a insertar
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección

Descripción

*El comando **ST SET PLAIN TEXT** inserta el texto pasado en el parámetro nuevTexto en el campo o la variable de texto con estilo designado por el parámetro objeto . Este comando aplica únicamente al texto plano del parámetro objeto, sin modificar las posibles etiquetas de estilo que contiene.*

*A diferencia del comando **ST SET TEXT**, **ST SET PLAIN TEXT** inserta únicamente texto plano. No debe pasar texto con etiquetas de estilo en nuevTexto. Si contiene los caracteres <, > o &, se consideran como caracteres estándar y se convierten en entidades HTML:*

- *'&' se convierte en &*
- *'<' se convierte en <*
- *'>' se convierte en >*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Durante la ejecución, si el objeto tiene el foco, el comando sólo se aplica al objeto que se está editando y no a su fuente de datos (campo o variable). Los cambios sólo se transfieren a la fuente (y a cualquier otro objeto con esta misma fuente) cuando el objeto que se está editando se valida, ya sea por perder el foco o con la tecla **Intro**. Si el objeto no tiene el foco, el comando se aplica directamente a la fuente de datos y los cambios son inmediatamente trasladados a otros objetos con la misma fuente.*
*Si se omite el parámetro *, indica que el parámetro objeto es un campo o una variable y se pasa una referencia de campo o variable en lugar de una cadena. El comando se aplica directamente al campo o a la variable y los cambios son transferidos inmediatamente a todos los objetos que utilizan esta fuente, incluyendo el objeto con el foco.*

En nuevTexto, pase el texto plano a insertar.

Los parámetros opcionales inicioSel y finSel le permiten designar una selección de texto en objeto. Los valores inicioSel y finSel dan una selección de texto plano, sin tener en cuenta las etiquetas de estilo encontradas en el texto. La acción del comando varía de acuerdo a los parámetros opcionales inicioSel y finSel:

- *Si omite inicioSel y finSel, **ST SET PLAIN TEXT** remplaza todo el texto de objeto por nuevTexto,*
- *Si sólo pasa inicioSel o si los valores de inicioSel y finSel son iguales, **ST SET PLAIN TEXT** inserta el texto nuevTexto en objeto comenzando en inicioSel,*
- *Si pasa ambos inicioSel y finSel, **ST SET PLAIN TEXT** remplaza el texto plano definido por estos límites por el texto nuevTexto.*
- *Puede pasar 0 en finSel para designar automáticamente el último carácter del texto (pase 1 en inicioSel para designar el primer carácter del texto).*

*4D ofrece constantes predefinidas que puede utilizar para designar automáticamente los límites de la selección en los parámetros inicioSel y finSel. Estas constantes están disponibles en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia de variable o de campo, el comando se aplica a todo el texto del objeto.

El estilo del primer carácter remplazado se utilizará para todo texto nuevTexto.

Si inicioSel es mayor que finSel, el texto no se modifica y la variable OK toma el valor 0 (excepto cuando el valor endSel es 0, ver arriba).

Ejemplo

Dada la siguiente variable texto multiestilo:

Please remember that the **X company has made a commitment** to you.

*Usted quiere insertar nombres de empresas guardadas en un campo texto. Estos nombres pueden contener, por ejemplo el carácter "&". En este caso, deberá utilizar el comando **ST SET PLAIN TEXT**:*

```
ST SET PLAIN TEXT(miTextoEstilo;[Empresa]Nombre;33;34)
```

Este es el resultado:

Please remember that the **Smith & Jones company has made a commitment** to you.

Este es el texto sin formato contenido en la variable:

```
Please remember that the <SPAN STYLE="font-weight:bold"> Smith &amp;amp; Jones </SPAN> <SPAN STYLE="font-weight:bold">company has made a commitment</SPAN> to you.
```

Puede constatar que el texto insertado se encapsuló dentro de un par de etiquetas de estilo adicionales. Estas etiquetas corresponden al estilo de los caracteres antes de la inserción. Este mecanismo permite garantizar una visualización correcta de los campos multiestilos en todos los casos.

Nota: *si utiliza el comando* ***ST SET TEXT*** *en este caso, 4D no inserta nada porque la presencia del carácter "&" no codificado evita la interpretación de las etiquetas de estilo presentes en la variable. Para mayor información, consulte la descripción de este comando.*

Variables y conjuntos del sistema

Después de la ejecución de este comando, la variable OK toma el valor 1 si no hay errores, de lo contrario toma el valor 0. Este es el caso en particular cuando las etiquetas de estilo no se evalúan correctamente (etiqueta incorrecta o falta).

En caso de un error, la variable no cambia. Cuando se produce un error en una variable cuando el texto está siendo evaluado, 4D transforma el texto en texto sin formato, como resultado, los caracteres <, > e & se convierten en entidades HTML.

ST SET TEXT

ST SET TEXT ({* ;} objeto ; nuevTexto {; inicioSel {; finSel}})

Parámetro	Tipo		Descripción
*	Operador	➡	Si se especifica, objeto es un nombre de objeto (cadena) Si se omite, objeto es una variable o un campo
objeto	Objeto de formulario	➡	Nombre del objeto (si se especifica *) o Variable o campo Texto (si se omite *)
nuevTexto	Texto	➡	Texto a insertar
inicioSel	Entero largo	➡	Inicio de la selección
finSel	Entero largo	➡	Fin de la selección

Descripción

*El comando **ST SET TEXT** inserta el texto pasado en el parámetro nuevText en el campo o variable de texto con estilo designado por el parámetro objeto. Este comando sólo se aplica al texto sin formato del parámetro de objeto, sin modificar las etiquetas de estilo que contiene. Se puede utilizar para modificar por programación, texto con estilo en la pantalla.*

*Si pasa el parámetro opcional *, indica que el parámetro objeto es un nombre de objeto (una cadena). Durante la ejecución, si el objeto tiene el foco, el comando sólo se aplica al objeto que se está editando y no a su fuente de datos (campo o variable). Los cambios sólo se transfieren a la fuente (y a cualquier otro objeto con esta misma fuente) cuando el objeto que se está editando se valida, ya sea por perder el foco o con la tecla **Intro**. Si el objeto no tiene el foco, el comando se aplica directamente a la fuente de datos y los cambios son inmediatamente trasladados a otros objetos con la misma fuente.*
*Si se omite el parámetro *, indica que el parámetro objeto es un campo o una variable y se pasa una referencia de campo o variable en lugar de una cadena. El comando se aplica directamente al campo o a la variable y los cambios son transferidos inmediatamente a todos los objetos que utilizan esta fuente, incluyendo el objeto con el foco.*

*En nuevText, pase el texto a insertar. El comando **ST SET TEXT** está diseñado para trabajar con texto enriquecido (multiestilo) con etiquetas de tipo . En todos los demás casos (particularmente, cuando trabaja con texto plano que contiene los caracteres <, > o &), debe utilizar el comando **ST SET PLAIN TEXT**. Si pasa texto plano con los caracteres <, > o & al comando **ST SET TEXT**, el comando no hace nada. Este principio de funcionamiento es necesario porque si inserta directamente una cadena como "a<b" dentro de un texto enriquecido, no distorsionará el análisis interno de las etiquetas . En este caso, el carácter "<" debe ser previamente codificado como "<", lo cual se puede hacer utilizando el comando **ST SET PLAIN TEXT** (ver también el ejemplo de este comando).*

Los parámetros opcionales inicioSel y finSel permiten designar una selección de texto en el objeto. Los valores inicioSel y finSel ofrecen una selección de texto sin formato, sin tener en cuenta ningún tipo de etiquetas de estilo en el texto. La acción del comando varía según los parámetros opcionales inicioSel y finSel:

- *Si omite inicioSel y finSel, **ST SET TEXT** reemplaza todo el texto de objeto por nuevText,*
- *Si pasa únicamente inicioSel o si los valores de inicioSel y finSel son iguales, **ST SET TEXT** inserta el texto nuevText en objeto a partir de inicioSel,*
- *Si pasa ambos inicioSel y finSel, **ST SET TEXT** remplaza el texto sin formato definido para estos límites con el texto nuevText.*
- *Puede pasar 0 en finSel para designar automáticamente el último carácter del texto (pase 1 en inicioSel para designar el primer caracter del texto).*

*4D ofrece constantes predefinidas de manera que pueda designar automáticamente los límites de selección en los parámetros inicioSel y finSel. Estas constantes se encuentran en el tema "**Texto multiestilo**":*

Constante	Tipo	Valor	Comentario
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto

(*) Debe pasar un nombre de objeto en objeto para poder utilizar esta constante. Si pasa una referencia a un campo o variable, el comando se aplica a todo el texto del objeto.

***Nota:** si inicioSel es superior a finSel, el texto no se modifica y la variable OK toma el valor 0 (excepto cuando finSel vale 0, ver arriba).*

Variables y conjuntos del sistema

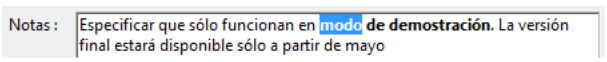
Después de ejecutar este comando, la variable OK toma el valor 1 si no se presenta ningún error; de lo contrario, toma el valor 0. Este es el caso particularmente cuando las etiquetas de estilo no se evalúan correctamente (etiquetas incorrectas o faltantes).

En caso de error, no cambia la variable. Cuando ocurre un error en una variable cuando se está evaluando el texto, 4D transforma el texto en texto plano; como resultado, los caracteres <, > y & se convierten en entidades HTML.

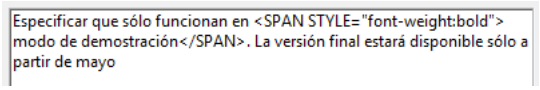
Ejemplo 1

Si quiere reemplazar el texto con estilo seleccionado por el usuario con el contenido de una variable.

Este es el texto seleccionado:



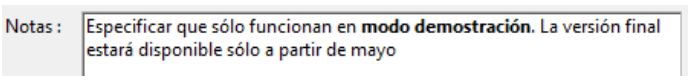
El contenido almacendo en el campo es el siguiente:

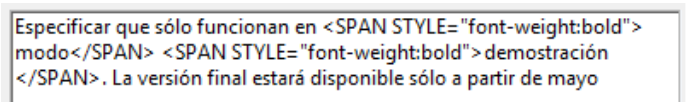


Después de la ejecución de este código:

```
vtempo:="Demonstración"
GET HIGHLIGHT([Productos]Notas;vInicio;vFin)
ST SET TEXT([Products]Notes;vtemp;vStart;vEnd)
```

El campo y su contenido son los siguientes:





Ejemplo 2

*Consulte el ejemplo del comando **ST SET PLAIN TEXT**.*

Transacciones

- *Utilización de transacciones*
- *Suspender las transacciones*
- *Active transaction*
- *CANCEL TRANSACTION*
- *In transaction*
- *RESUME TRANSACTION*
- *START TRANSACTION*
- *SUSPEND TRANSACTION*
- *Transaction level*
- *VALIDATE TRANSACTION*

Utilización de transacciones

Las transacciones son una serie de modificaciones efectuadas al interior de un proceso sobre datos relacionados. Una transacción no se guarda permanente en la base hasta que la transacción sea validada. Si no se completa una transacción, bien sea porque se cancela o por un evento externo, las modificaciones no se guardan.

Durante una transacción, todos los cambios realizados a los datos de la base dentro del proceso se almacenan localmente en un buffer temporal. Si la transacción se acepta con ***VALIDATE TRANSACTION****, los cambios se guardan de manera permanente. Si la transacción se cancela con* ***CANCEL TRANSACTION****, los cambios no se guardan. En todos los casos, ni la selección actual ni el registro actual son modificados por los comandos de gestión de transacciones.*

4D soporta transacciones anidadas, es decir, transacciones en varios niveles jerárquicos. El número de subtransacciones autorizadas es ilimitado. El comando ***Transaction level*** *permite conocer el nivel actual de transacción en el cual se ejecuta el código.*

Cuando utiliza transacciones anidadas, el resultado de cada subtransacción depende de la validación o cancelación de la transacción de nivel superior. Si se valida la transacción de nivel superior, los resultados de las subtransacciones se confirman (validación o cancelación). Por el contrario, si la transacción superior se cancela, todas las subtransacciones se cancelan, sin importar sus resultados.
Nota: *por razones de compatibilidad, las transacciones anidadas están desactivadas por defecto en las bases de datos convertidas de versiones anteriores a la v11 (ver* ***Página Compatibilidad****).*

4D incluye una funcionalidad que le permite suspender y reanudar las transacciones dentro de su código 4D. Cuando se suspende una transacción, puede ejecutar operaciones de forma independiente de la transacción misma y luego reanudar la transacción para validarla o cancelarla como de costumbre. Cuando se suspende la transacción, puede en particular:

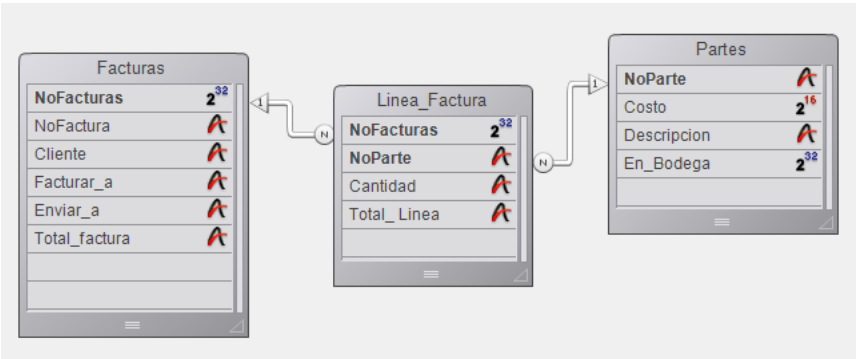
- *crear o modificar registros fuera de la transacción (por ejemplo, para incrementar un contador de números de facturas),*
- *iniciar, suspender y/o cerrar otras transacciones.*

Para obtener más información sobre este punto, consulte ***Suspender las transacciones****.*

Ejemplos de transacciones

En este ejemplo, la base es una sistema de facturación simple. Las líneas de las facturas son almacenadas en una tabla llamada [Linea_Factura], la cual está relacionada con la tabla [Facturas] por una relación entre los campos [Facturas]NoFacturas y [Linea_Factura]NoFacturas. Cuando se añade una factura, se calcula un número único, utilizando el comando ***Sequence number****. La relación entre [Facturas] y [Linea_Factura] es una relación automática Muchos a Uno. La casilla de selección Autoasignar valor en el subformulario está seleccionada.*

La relación entre [Linea_Factura] y [Partes] es manual.



Cuando un usuario introduce una factura, se ejecutan las siguientes acciones:

- *Adición de un registro en la tabla [Facturas].*
- *Adición de varios registros en la tabla [Linea_Factura].*
- *Actualización del campo [Partes]En_Bodega de cada parte listada en la factura.*

Este ejemplo es una situación típica en la cual necesita utilizar una transacción. Debe asegurarse de poder guardar todos estos registros durante la operación o de que poder cancelar la transacción si un registro no puede añadirse o actualizarse. En otras palabras, debe guardar los datos relacionados.

Si no utiliza una transacción, no puede garantizar la integridad de datos lógica de su base. Por ejemplo, si un registro de la tabla [Partes] está bloqueado, no podrá actualizar la cantidad almacenada en el campo [Partes]En_Bodega. Este campo se volverá entonces lógicamente incorrecto. La suma de las partes vendidas y restantes en bodega no será igual a la cantidad original introducida en el registro. Puede evitar tal situación utilizando las transacciones.

Hay varias maneras de introducir datos utilizando transacciones:

1. Puede administrar las transacciones utilizando los comandos de transacción ***START TRANSACTION****,* ***VALIDATE TRANSACTION****, y* ***CANCEL TRANSACTION****. Puede escribir, por ejemplo:*

```
READ WRITE([Linea_Factura])
READ WRITE([Partes])
FORM SET INPUT([Facturas];"Entrada")
Repeat
```

```
    START TRANSACTION
    ADD RECORD([Facturas])
    If(OK=1)
        VALIDATE TRANSACTION
    Else
        CANCEL TRANSACTION
    End if
Until(OK=0)
READ ONLY(*)
```

*2. Para reducir los bloqueos de registros durante la entrada de datos, puede también seleccionar administrar transacciones a partir del método de formulario y acceder a las tablas en **READ WRITE** únicamente cuando sea necesario.*

Usted efectúa la entrada de datos utilizando el formulario de entrada para [Facturas], el cual contiene la tabla relacionada [Facturas]Lineas en un subformulario. El formulario tiene dos botones: bCancel y bOK, ambos son botones sin acciones.

El bucle de adición se convierte en:

```
READ WRITE([Linea_Factura])
READ ONLY([Partes])
FORM SET INPUT([Facturas];"Entrada")
Repeat
    ADD RECORD([Facturas])
Until(bOK=0)
READ ONLY([Linea_Factura])
```

Note que la tabla [Partes] está en modo de acceso "sólo lectura" durante la entrada de datos. El acceso lectura/escritura estará disponible únicamente si los datos se validan.

La transacción se abre en el método de formulario de entrada de la tabla [Facturas]:

```
Case of
    :(Form event=On Load)
        START TRANSACTION
        [Facturas ]NoFacturas:=Sequence number([Facturas ]NoFacturas)
    Else
        [Facturas]Total_Factura:=Sum([Linea_Factura]Total_Linea)
End case
```

Si hace clic en el botón bCancel, la entrada y la transacción deben ser canceladas.

Este es el método de objeto del botón bCancel:

```
Case of
    :(Form event=On Clicked)
        CANCEL TRANSACTION
        CANCEL
End case
```

Si hace clic en el botón bOK, la entrada y la transacción deben ser aceptadas. Este es el método de objeto del botón bOK:

```
Case of
    :(Form event=On Clicked)
        $NbLineas:=Records in selection([Linea_Factura])
        READ WRITE([Partes]) ` Cambiar a acceso lectura/escritura para la tabla [Partes]
        FIRST RECORD([Linea_Factura]) ` Iniciar en la primera línea
        $ValidTrans:=True ` Asume que todo estará OK
        For($Line;1;$NbLines) ` Para cada línea
            RELATE ONE([Linea_Factura]NoParte)
            OK:=1 ` Asume que usted quiere continuar
            While(Locked([Partes]) & (OK=1)) ` Tratar de obtener el registro en acceso Lectura/Escritura
                CONFIRM("La parte "+[Linea_Factura]NoParte+" está en uso. ¿Espera?")
                If(OK=1)
                    DELAY PROCESS(Current process;60)
                    LOAD RECORD([Partes])
                End if
            End while
            If(OK=1)
 ` Actualizar cantidad en la bodega
                [Partes]En Bodega:=[Partes]En Bodega-[Linea_Factura]Cantidad
                SAVE RECORD([Partes]) ` Guardar registro
            Else
                $Linea:=$NbLineas+1 ` Dejar el bucle
                $ValidTrans:=False
            End if
            NEXT RECORD([Linea_Factura]) ` Ir a la siguiente línea
```

```
        End for
        READ ONLY([Partes]) ` Definir el estado de la tabla en modo sólo lectura
        If($ValidTrans)
            SAVE RECORD([Facturas]) ` Guarda el registro Facturas
            VALIDATE TRANSACTION ` Validar todas las modificaciones a la base
        Else
            CANCEL TRANSACTION ` Cancelar todo
        End if
        CANCEL ` Dejar el formulario
    End case
```

*En este código, llamamos al comando **CANCEL** sin importar en cual botón el usuario haga clic. El nuevo registro no está validado por una llamada a **ACCEPT**, pero por el comando **SAVE RECORD**. Además, note que **SAVE RECORD** se llama justo antes que el comando **VALIDATE TRANSACTION**. Por lo tanto, guardar el registro [Facturas] es en realidad parte de la transacción. Llamar el comando **ACCEPT** también validaría el registro, pero en este caso la transacción será validada antes de que el registro [Facturas] se guarde. En otras palabras, el registro sería guardado fuera de la transacción.*

Dependiendo de sus necesidades, puede personalizar su base, como se muestra en estos ejemplos. En el último ejemplo, la gestión de bloqueo de registros de la tabla [Partes] puede desarrollarse más.

Suspender las transacciones

Principio

Suspender una transacción es útil cuando se necesita, dentro de una transacción, realizar ciertas operaciones que no necesitan ser ejecutadas bajo el control de esta transacción. Por ejemplo, imaginemos el caso de que un cliente hace un pedido, dentro de una transacción y también actualiza su dirección. A continuación, el cliente cambia de opinión y se cancela la orden. La transacción se cancela, pero usted no quiere que el cambio de dirección se revierta. Este es un ejemplo típico en el que suspender la transacción es útil. Tres comandos se utilizan para suspender y reanudar las transacciones:

- ***SUSPEND TRANSACTION**: detiene la transacción actual. Todos los registros actualizados o añadidos permanecen bloqueados.*
- ***RESUME TRANSACTION**: reactiva una transacción suspendida.*
- ***Active transaction**: devuelve **False** si la transacción se suspende o si no hay ninguna transacción actual y **True** si se inicia o reactiva.*

Ejemplo

Este ejemplo ilustra la necesidad de suspender una transacción. En una base de datos de facturas, queremos obtener un nuevo número de la factura durante una transacción. Este número se calcula y se almacena en una tabla [Settings]. En un entorno multiusuario, los accesos concurrentes deben estar protegidos; Sin embargo, por la transacción, la tabla [Settings] podría ser bloqueada por otro usuario a pesar de que estos datos sean independientes de la transacción principal. En este caso, puede suspender la transacción cuando se accede a la tabla.

```
//Método estándar que crea una factura
START TRANSACTION
...
CREATE RECORD([Invoices])
[Invoices]InvoiceID:=GetInvoiceNum //llama al método para obtener un número disponible
...
SAVE RECORD([Invoices])
VALIDATE TRANSACTION
```

*El método **GetInvoiceNum** suspende la transacción antes de ejecutarla. Note que este código funcionará incluso si el método se llama desde fuera de una transacción:*

```
//Método proyecto GetInvoiceNum
//GetInvoiceNum -> Siguiente número de factura disponible
C_LONGINT($0)
SUSPEND TRANSACTION
ALL RECORDS([Settings])
If(Locked([Settings])) //Acceso multi-usuario
   While(Locked([Settings]))
      MESSAGE("Waiting for locked Settings record")
      DELAY PROCESS(Current process;30)
      LOAD RECORD([Settings])
   End while
End if
[Settings]InvoiceNum:=[Settings]InvoiceNum+1
$0:=[Settings]InvoiceNum
SAVE RECORD([Settings])
UNLOAD RECORD([Settings])
RESUME TRANSACTION
```

Funcionamiento detallado

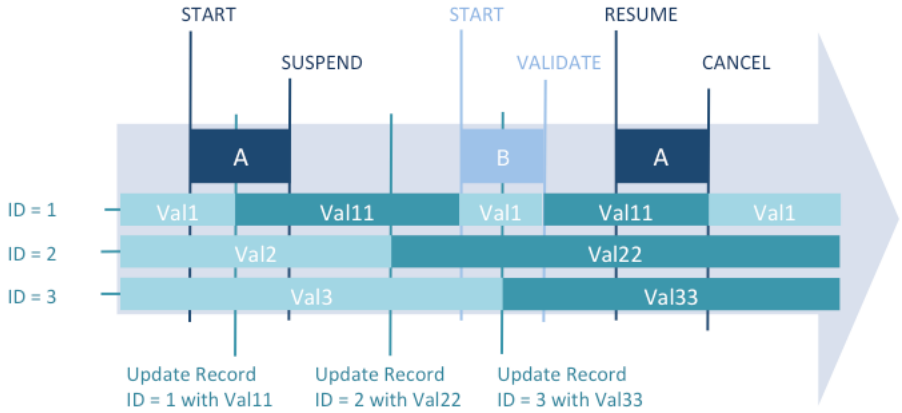
¿Cómo suspender una transacción?

Cuando se suspende una transacción, se aplican los siguientes principios:

- *Puede acceder a los registros que se han añadido o modificado durante la transacción y no puede acceder a los registros que se han eliminado durante la transacción.*
- *Puede crear, guardar, eliminar o modificar los registros fuera de la transacción.*
- *Puede iniciar una nueva transacción, pero dentro de esta transacción incluida, no podrá ver los registros o los valores de registros que se han añadido o modificado durante la transacción suspendida. De hecho, esta nueva transacción es totalmente independiente de la que se ha suspendido, similar a una transacción de otro proceso y ya que la transacción suspendida más tarde podría reanudarse o cancelarse, cualquier registro añadido o modificado se oculta automáticamente para la nueva transacción. Tan pronto como confirme o cancele la nueva transacción, puede ver estos registros de nuevo.*
- *Todos los registros que son modificados, eliminados o añadidos dentro de la transacción suspendida permanecen bloqueados para otros procesos. Si intenta modificar o eliminar estos registros fuera de la transacción o en una nueva*

transacción, se genera un error.

Estas implementaciones se resumen en el siguiente gráfico:



Los valores editados durante la transacción A (registro ID1 toma el valor Val11) no están disponibles en una nueva transacción (B) creada durante el período "suspendido". Los valores editados durante el período de suspensión (registro ID2 toma el valor Val22 y registro ID3 toma el valor Val33), se conservan después de que se cancela la transacción A.

Las funcionalidades específicas se han añadido para manejar los errores:

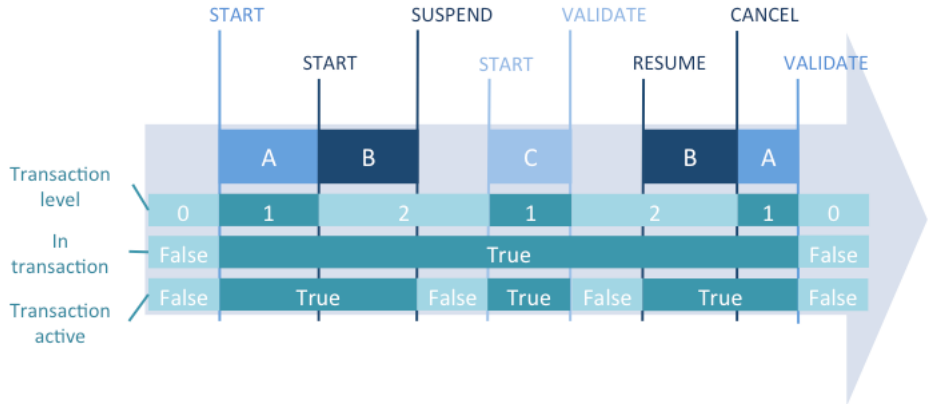
- *El registro actual de cada tabla se bloquea temporalmente si se modifica durante la transacción y se desbloquea automáticamente cuando se reanuda la operación. Este mecanismo es importante para evitar que partes de la transacción se guarden de forma inesperada.*
- *Si se ejecuta una secuencia inválida tal como, iniciar transacción / suspender transacción / iniciar transacción / reactivar transacción, se genera un error. Este mecanismo evita que los desarrolladores olviden confirmar o cancelar las sub transacciones incluidas antes de reanudar la operación suspendida.*

Suspender y reactivar transacciones

*El comando existente **In transaction** devuelve **True** cuando se ha iniciado una transacción, incluso si está suspendida. Para saber si se suspende la transacción actual, es necesario utilizar el nuevo comando **Transacción activa**, que devuelve **False** en este caso.*

*Ambos comandos, sin embargo, también devuelven **False** si no se ha iniciado ninguna transacción. Después puede ser necesario utilizar el comando **Transaction level**, que devuelve 0 en este contexto (ninguna operación iniciada).*

El siguiente gráfico ilustra los diversos contextos de transacción y los valores correspondientes devuelto por los comandos de transacción:



Active transaction

Active transaction -> Resultado

Parámetro	Tipo		Descripción
Resultado	Booleano		Devuelve False si se suspende la transacción actual

Descripción

*El comando **Active transaction** devuelve **True** si el proceso actual está en transacción y si esta transacción no se suspende. Devuelve **False** si no hay una transacción en curso, o si se suspende la transacción actual. Una transacción puede ser suspendida utilizando el comando **SUSPEND TRANSACTION**.*

*Dado que el comando también devolverá **False** si el proceso actual no está en transacción, puede que necesite utilizar el comando **In transaction** para saber si el proceso está en transacción.*

Para más información, consulte el párrafo .

Descripción

Usted quiere conocer el estado de la transacción actual:

```
If(In transaction)
    If(Not(Active transaction))
        ALERT("La transacción actual está suspendida")
    Else
        ALERT("La transacción actual está activa")
    End if
Else
    ALERT("No estamos en transacción")
End if
```

CANCEL TRANSACTION

CANCEL TRANSACTION

Este comando no requiere parámetros

Descripción

CANCEL TRANSACTION *anula la transacción abierta por el comando* ***START TRANSACTION*** *del nivel correspondiente en el proceso actual.* ***CANCEL TRANSACTION*** *anula todas las operaciones que llegaran a ejecutarse en los datos durante la transacción.*

Nota: CANCEL TRANSACTION *no tiene efecto en las posibles modificaciones efectuadas en los registros actuales que no se guardaron, se siguen mostrando después de la ejecución del comando.*

In transaction

In transaction -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Booleano*		*Devuelve TRUE si el proceso actual está en transacción*

Descripción

*El comando **In transaction** devuelve **True** si el proceso actual está en transacción, de lo contrario devuelve **False**.*

Ejemplo

Si efectúa las operaciones (adición, modificación, o eliminación de registros) con múltiples registros, puede encontrarse con registros bloqueados. En este caso, para preservar la integridad de los datos, debe tener abierta una transacción, de manera que pueda "devolver" toda la operación y dejar la base intacta.

*Si efectúa la operación desde un trigger o una subrutina que puede ser llamado(a) en una transacción o fuera de transacción, la utilización del comando **In transaction** permite verificar que el método del proceso actual o el método llamante abrió bien una transacción. Si no es el caso, no comienza la transacción, porque en caso de una falla en el proceso, no podría deshacer las operaciones efectuadas.*

RESUME TRANSACTION

RESUME TRANSACTION
Este comando no requiere parámetros

Descripción

*El comando **RESUME TRANSACTION** reactiva la transacción que se suspendió utilizando **ServerSpecialBuild** en el nivel correspondiente en el proceso actual. Todas las operaciones que se ejecutan después de este comando se llevan a cabo bajo el control de transacciones (excepto cuando varias transacciones suspendidas están anidadas).*
*Para más información, consulte la sección **Suspender las transacciones**.*

START TRANSACTION

START TRANSACTION
Este comando no requiere parámetros

Descripción

START TRANSACTION *inicia una transacción en el proceso actual. Todos los cambios a los datos (registros) de la base dentro de la transacción se almacenan temporalmente hasta que la transacción sea validada o cancelada.*

A partir de la versión 11 de 4D, puede anidar varias transacciones (subtransaccciones). Cada transacción o subtransaction debe ser finalmente cancelada o validada. Note que si la transacción principal se cancela, todas las subtransacciones también se cancelan, sin importar su resultado.

SUSPEND TRANSACTION

SUSPEND TRANSACTION

Este comando no requiere parámetros

Descripción

*El comando **SUSPEND TRANSACTION** suspende la transacción actual en el proceso actual. A continuación, puede manipular los datos en otras partes de la base, por ejemplo, sin que sean incluidos en la transacción, y al mismo tiempo preservar el contexto actual de la transacción. Todos los registros que han sido actualizados o añadidos en la transacción están bloqueados hasta que la transacción se reactiva con el comando **Versioning / Client**.*

*Para más información, por favor consulte la sección **Suspender las transacciones**.*

Transaction level

Transaction level -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entero largo*		*Nivel de transacción actual (0 si ninguna transacción se ha iniciado)*

Descripción

*El comando **Transaction level** devuelve el nivel de transacción actual para el proceso. Este comando tiene en cuenta todas las transacciones del proceso actual, sin importar si fueron iniciadas vía el lenguaje 4D o vía SQL.*

VALIDATE TRANSACTION

VALIDATE TRANSACTION
Este comando no requiere parámetros

Descripción

VALIDATE TRANSACTION *acepta la transacción abierta por el comando* ***START TRANSACTION*** *del nivel correspondiente en el proceso actual.* ***VALIDATE TRANSACTION*** *guarda todas las modificaciones efectuadas a los datos de la base que ocurrieron durante la transacción.*

A partir de la versión 11 de 4D, puede anidar varias transacciones (subtransacciones). Si la transacción principal se cancela, todas las subtransacciones se cancelan, incluso si han sido validadas individualmente utilizando este comando.

Variables y conjuntos del sistema

La variable sistema OK toma el valor 1 si la transacción se ha validado correctamente; de lo contrario, toma el valor 0.

Tenga en cuenta que cuando OK toma el valor 0, la transacción automáticamente se cancela internamente (equivalente a ***CANCEL TRANSACTION****). Por lo tanto, no debe llamar explícitamente* ***CANCEL TRANSACTION*** *si OK=0, particularmente en el contexto de las transacciones anidadas, ya que la cancelación se aplicará entonces a la transacción de nivel superior.*

Triggers

Triggers
Trigger event
Trigger level
TRIGGER PROPERTIES

Triggers

Un trigger es un método asociado a una tabla. Es una propiedad de una tabla. Usted no llama a los triggers; los triggers son llamados automáticamente por el motor de 4D cada vez que manipula registros de la tabla (adición, eliminación, modificación, y carga). Puede escribir triggers muy simples, y luego volverlos más sofisticados.

Los triggers pueden evitar operaciones "ilegales" en los registros de su base. Son una herramienta muy poderosa que permite controlar las operaciones en tablas, como también evitar perdidas de datos accidentales. Por ejemplo, en un sistema de facturación, puede evitar que un usuario cree una factura sin especificar el cliente al que debe facturarse.

Nota: *ver 4D Security guide para una visión general de las funcionalidades de seguridad de 4D.*

Activar y crear un trigger

Por defecto, cuando crea una tabla en el entorno Diseño, la tabla no tiene trigger.

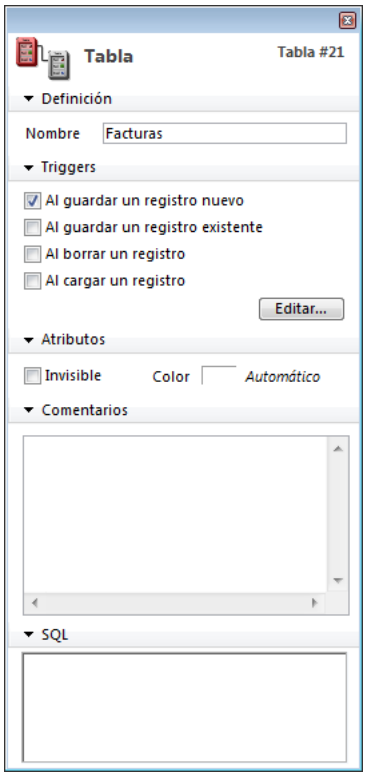
Para utilizar un trigger para una tabla, necesita:

- *Activar el trigger e indicar a 4D cuando llamarlo.*
- *Escribir el código para el trigger.*

Activar un trigger que no está escrito o escribir un trigger sin activarlo no afecta las operaciones efectuadas en una tabla.

1. Activar un Trigger

Para activar un trigger, seleccione las opciones ***Triggers*** *para la tabla en la ventana de propiedades de la tabla:*



Al guardar un registro existente

Si se selecciona esta opción, el trigger se llamará cada vez que se modifique un registro de la tabla.
Esto sucede cuando:

- *se modifica un registro en la entrada de datos (entorno Diseño, comando* ***MODIFY RECORD*** *o el comando SQL* ***UPDATE****).*
- *se guarda un registro existente utilizando* ***SAVE RECORD****.*
- *se llama un comando que guarda registros existentes (por ejemplo,* ***ARRAY TO SELECTION****,* ***APPLY TO SELECTION****, etc.).*
- *se utiliza un plug-in que llama al comando* ***SAVE RECORD****.*

Nota: *por razones de optimización, el trigger no se llama cuando el registro es guardado por el usuario o vía el comando* ***SAVE RECORD*** *si ningún campo de la tabla ha sido modificado en el registro. Si quiere "forzar" el llamado del trigger en este caso, simplemente puede asignar un campo:*

```
[latabla]elcampo:=[latabla]elcampo
```

Al borrar un registro

Si selecciona esta opción, el trigger se llamará cada vez que se borre un registro de la tabla.
Esto sucede cuando:

- *Cuando se borra un registro (entorno Diseño o llamando a los comandos **DELETE RECORD**, **DELETE SELECTION** o al comando SQL **DELETE**).*
- *Cuando se efectúan operaciones que provocan la eliminación de un registro relacionado por intermedio de las opciones de control de eliminación de una relación*
- *Cuando se utiliza un plug-in que llama al comando **DELETE RECORD**.*

***Nota:** el comando **TRUNCATE TABLE** no llama al trigger.*

Al guardar un nuevo registro

Si esta opción se selecciona, el trigger se llamará cada vez que un registro se cree en la tabla, es decir en las siguientes circunstancias:

- *Cuando se añade un registro en la entrada de datos (entorno Diseño, comando **ADD RECORD** o comando **INSERT**).*
- *Cuando se crea y guarda un registro con **CREATE RECORD** y **SAVE RECORD**. Note que el trigger se llama en el momento en que llama **SAVE RECORD**, no cuando se crea.*
- *Cuando se importan registros (entorno Diseño o utilizando un comando de importación).*
- *Cuando se llaman otros comandos que crean y/o guardan nuevos registros (por ejemplo, **ARRAY TO SELECTION**, **SAVE RELATED ONE**, etc.).*
- *Cuando se utiliza un plug-in que llama los comandos **CREATE RECORD** y **SAVE RECORD**.*

2. Crear un trigger

*Para crear un trigger para una tabla, utilice la ventana de propiedades de la tabla, haga clic en el botón **Editar** o presione Alt (en Windows) u Opción (Macintosh) y doble clic en la tabla en la ventana de estructura. Para mayor información, consulte el manual de Diseño de 4D.*

Eventos de base

*Un trigger puede ser llamado por uno de los cuatro **eventos de base** descritos anteriormente. En el trigger, puede detectar qué evento está ocurriendo llamando la función **Trigger event**. Esta función devuelve un valor numérico que indica el evento de base.*

*Generalmente, se escribe un trigger con una estructura de tipo **Case of** sobre el resultado devuelto por **Trigger event**. Puede utilizar las constantes del tema **_o_LAST SUBRECORD***

```
 //  Trigger for [unaTabla]
C_LONGINT($0)
$0:=0 // Asume que la petición será aceptada
Case of
   :(Trigger event=On Saving New Record Event)
 // Realizar las acciones apropiadas para guardar el nuevo registro creado
   :(Trigger event=On Saving Existing Record Event)
 // Realizar las acciones apropiadas para guardar un registro existente
   :(Trigger event=On Deleting Record Event)
 // Realizar las acciones apropiadas para borrar un registro
End case
```

Los triggers son funciones

Un trigger tiene dos propósitos:

- *Efectuar acciones sobre el registro justo antes de que se guarde o borre, o justo después de ser cargado.*
- *Aceptar o rechazar una operación de base de datos.*

1. Efectuar las acciones

Cada vez que se guarda un registro (añadido o modificado) a una tabla [Documentos], usted quiere "marcar" el registro con los marcadores de creación y modificación. Puede escribir el siguiente trigger:

```
 // Trigger for table [Documentos]
Case of
   :(Trigger event=On Saving New Record Event)
      [Documentos]Creacion_marca:=Time stamp
      [Documentos]Modificacion_marca:=Time stamp
   :(Trigger event=On Saving Existing Record Event)
      [Documentos]Modificacion_marca:=Time stamp
End case
```

***Nota**: la función Time stamp utilizada en este ejemplo es un pequeño método de proyecto que devuelve el número de segundos transcurridos desde una fecha elegida arbitrariamente.*

Una vez este trigger ha sido escrito y activado, no importa de que manera añada o modifique un registro en la tabla de la tabla [Documentos] (entrada de datos, importación, método de proyecto, plug-in 4D), los campos [Documentos]Creacion_marca y [Documentos]Modificacion_marca serán asignados automáticamente por el trigger antes de que el registro se escriba en el disco.

***Nota**: ver el ejemplo del comando #cmd id="477"/] para un análisis completo de este ejemplo.*

2. Aceptar o rechazar la operación de una base

*Para aceptar o rechazar una operación de la base, el trigger debe devolver un **código de error de trigger** en el resultado de la función $0.*

Ejemplo

Tomemos el caso de una tabla [Empleados]. Durante la entrada de datos, usted controla el campo [Empleados]Numero_Seguridad_Social. Cuando el usuario hace clic en el botón de validación, usted verifica el campo utilizando el método de objeto del botón:

```
// Método de objeto bAccept
If(Good SS number([Empleados]Numero_Seguridad_Social))
   ACCEPT
Else
   BEEP
   ALERT("Introduzca un número de seguridad social y haga clic de nuevo en OK.")
End if
```

Si el valor del campo es correcto, acepta la entrada de datos; si el valor del campo no es correcto, muestra una alerta y permanece en entrada de datos.

Si también crea registros para la tabla [Empleados] por programación, el siguiente código sería válido pero violaría la regla expresada en el método de objeto creado anteriormente:

```
// Extracción de un método de proyecto
// ...
CREATE RECORD([Empleados])
[Empleados]Name:="DOE"
SAVE RECORD([Empleados]) // <-- ¡Violación de la regla! El número de seguridad social no ha sido asignado
// ...
```

Utilizando un trigger para la tabla [Empleados], puede implementar la regla [Empleados]Numero_Seguridad_Social en todos los niveles de la base. El trigger se vería así:

```
// Trigger for [Empleados]
$0:=0
$dbEvent:=Trigger event
Case of
   :(($dbEvent=On Saving New Record Event)|($dbEvent=On Saving Existing Record Event))
      If(Not(Good SS number([Empleados]Numero_Seguridad_Social)))
         $0:=-15050
      Else
// ...
      End if
// ...
End case
```

Una vez este trigger está escrito y activado, la línea ***SAVE RECORD*** *([Empleados]) generará un error de base -15050, y el registro NO se guardará.*

De la misma forma, si un plug-in 4D intenta guardar un registro en [Empleados] con un número de seguridad social incorrecto, el trigger generará el mismo error y el registro no se guardará.

El trigger garantiza que nadie (usuario, desarrollador, plug-in, 4D Open client con 4D Server) pueda violar la regla del número de seguridad social, bien sea deliberada o accidentalmente.

Note que incluso si no tiene un trigger para una tabla, la base puede devolver errores cuando se trata de guardar o borrar un registro. Por ejemplo, si intenta guardar un registro con un valor duplicado en un campo indexado único, se devuelve el error -9998.

Los triggers devuelven nuevos tipos de errores en 4D:

- *4D administra los errores "normales": índice único, control de datos relacionales, etc.*
- *Utilizando triggers, puede crear códigos de errores únicos para su aplicación.*

***Importante**: puede devolver el código de error de su elección. Sin embargo, NO utilice códigos de error utilizados por el motor de 4D. Recomendamos utilizar códigos de error entre -32 000 y -15 000. Nos reservamos los errores superiores a -15 000 para el motor de 4D.*

A nivel del proceso, usted administra los errores trigger de la misma manera que los errores de motor de base de datos:

- *Puede permitir a 4D mostrar la caja de diálogo de error estándar, luego se interrumpe el método.*
- *Puede utilizar un método de gestión de errores instalado utilizando* ***ON ERR CALL*** *y recuperar el error de la manera apropiada.*

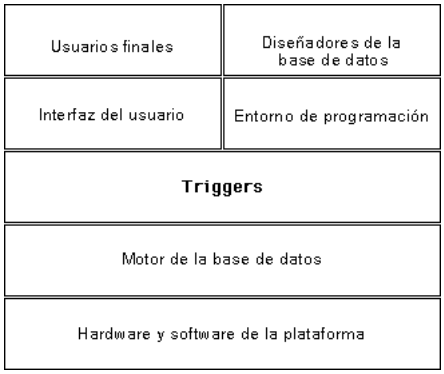
***Notas**:*

- *Durante la entrada de datos, si un error trigger es devuelto mientras intenta validar o borrar un registro, el error se trata como un error de índice único. La caja de diálogo de error se muestra, y permanece en la entrada de datos. Incluso si utiliza una base en el entorno Diseño (no en el entorno Aplicación), usted se beneficia del uso de triggers.*
- *Cuando se genera un error por un trigger en el marco de un comando que actúa sobre una selección de registros (como* ***DELETE SELECTION****), la ejecución del comando se detiene inmediatamente, sin que la selección haya sido procesada completamente. Este caso requiere una gestión apropiada por parte del desarrollador, basada, por ejemplo, en la conservación temporal de la selección, el procesamiento y la eliminación del error antes de la ejecución del trigger, etc.*

Incluso cuando un trigger no devuelve un error (`$0:=0`), esto no significa que una operación de la base será exitosa, puede ocurrir una violación de índice único. Si la operación es la actualización de un registro, el registro puede estar bloqueado, se puede producir un error de entrada/salida puede ocurrir, etc. Estas verificaciones son efectuadas después de la ejecución del trigger. Sin embargo, desde el punto de vista del nivel superior del proceso en ejecución, los errores devueltos por el motor de la base de datos o por un trigger son de la misma naturaleza, un error trigger es un error del motor de la base de datos.

Triggers y la arquitectura 4D

Los triggers funcionan al nivel del motor de la base de datos. Este punto se resume en el siguiente diagrama:



Los triggers se ejecutan en el equipo donde está el motor de la base de datos. Esto es evidente en el caso de 4D en local. En 4D Server, los triggers se ejecutan en el equipo servidor (en el proceso activo) y no en el equipo cliente.

Cuando se llama un trigger, se ejecuta dentro del contexto del proceso que intenta la operación. Este proceso, invoca la ejecución del trigger y se llama ***proceso llamante****.*

Los elementos incluidos en este contexto difieren si la base se ejecuta con 4D en modo local o con 4D Server:

- *Con 4D en modo local, el trigger funciona con las selecciones actuales, registros actuales, estados de lectura/escritura de las tablas, operaciones de bloqueos de registros, etc., del proceso llamante.*
- *Con 4D Server, sólo el contexto de base de datos del proceso cliente llamante se conserva (bloqueo de registros, estados transaccionales).*
  4D Server igualmente garantiza que el registro actual de la tabla del trigger esté correctamente posicionado.
  Los otros elementos contextuales (selecciones actuales por ejemplo) son las del proceso del trigger.

Tenga cuidado cuando utilice otros objetos de la base y del lenguaje del entorno 4D, porque un trigger podría ejecutarse en una maquina diferente de la del proceso que lo llama ¡Este es el caso con 4D Server!

- ***Variables interproceso****: un trigger tiene acceso a las variables interproceso del equipo donde se ejecuta. Con 4D Server, puede acceder a una maquina diferente a la del proceso llamante.*
- ***Variables proceso****: cada trigger tiene su propia tabla de variables proceso. Un trigger no tiene acceso a las variables proceso del proceso llamante.*
- ***Variables locales****: puede utilizar las variables locales en un trigger. Su alcance es la ejecución del trigger; se crean/eliminan en cada ejecución.*
- ***Semáforos****: un trigger puede probar o fijar semáforos globales y locales (en el equipo donde se ejecuta). Sin embargo, un trigger debe ejecutar rápidamente, de manera que debe ser muy cuidadoso cuando pruebe o defina semáforos dentro de triggers.*
- ***Conjuntos y selecciones temporales****: si utiliza un conjunto o una selección temporal en un trigger, trabaja en el equipo donde los triggers se ejecutan.*
- ***Interfaz del usuario****: NO utilice elementos de la interfaz del usuario en un trigger (alertas, mensajes o cajas de diálogo). De la misma forma, debe limitar todo seguimiento de triggers en la ventana del* ***Depurador****. Recuerde que en Cliente/Servidor, los triggers se ejecutan en el equipo 4D Server. Un mensaje de alerta en el equipo servidor no ayuda al usuario en un equipo cliente. Deje al proceso llamante administrar la interfaz del usuario.*

Tenga en cuenta que si utiliza el sistema de contraseñas de 4D, puede ejecutar el comando ***Current user*** *en el trigger con el fin, por ejemplo, de guardar el nombre del usuario en el origen de la llamada del trigger en una tabla con historial, incluso en modo cliente-servidor.*

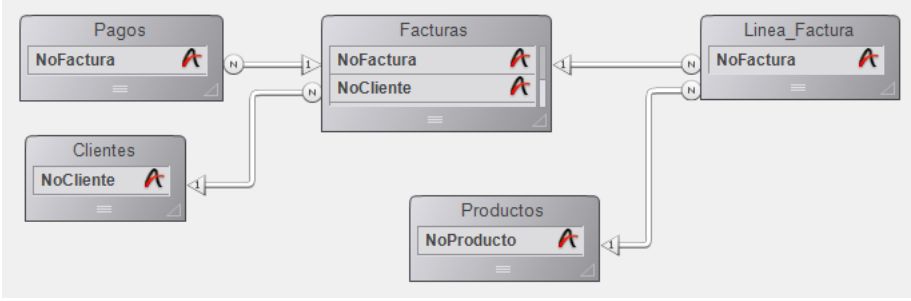
Triggers y transacciones

Las transacciones deben administrarse en el nivel del proceso llamante. No deben administrarse a nivel del trigger. Si durante la ejecución del trigger, tiene que añadir, modificar o borrar varios registros, primero debe utilizar el comando ***In transaction*** *desde el trigger para probar si el proceso llamante está en transacción actualmente. Si no es el caso, el trigger podría encontrarse con un registro bloqueado. Por lo tanto, si el proceso llamante no está en transacción, no comienzan las operaciones en los registros y devuelve un error en `$0` para indicar al proceso llamante que la operación de la base de datos debe ejecutarse en una transacción. Por otra parte, si encuentra registros bloqueados, el proceso llamante no podrá deshacer las acciones del trigger.*

Nota*: con el fin de optimizar el funcionamiento combinado de los triggers y transacciones, 4D no llama triggers después de la ejecución de* ***VALIDATE TRANSACTION****. Esto evita que los triggers se ejecuten dos veces.*

Triggers en cascada

Dada la siguiente estructura de ejemplo:



***Nota**: las tablas han sido contraídas; tienen más campos de los que se muestran.*

Supongamos que la base de datos "autoriza" la eliminación de una factura. Podemos examinar cómo sería tratada tal operación a nivel del trigger (porque también podría realizar eliminaciones a nivel del proceso).

Para conservar la integridad relacional de los datos, la eliminación de una factura requiere las siguientes acciones de parte del trigger de [Facturas]:

- *Disminuir el campo Ventas de la tabla [Clientes], en la cantidad de la factura.*
- *Borrar todos los registros de [Linea_Factura] relacionados con la factura.*
- *Esto también implica que el trigger de [Linea_Factura] disminuya el campo Cantidad vendida de los registros [Productos] relacionados con la línea de factura a eliminar.*
- *Borrar todos los registros de [Pagos] relacionados con la factura borrada.*

Primero, el trigger de [Facturas] debe efectuar estas acciones sólo si el proceso llamante está en transacción, de manera que sea posible deshacer en caso de encontrar un registro bloqueado.

*Segundo, el trigger de [Linea_Factura] está en **cascada** con el trigger de [Facturas]. El trigger [Linea_Factura] se ejecuta dentro de la ejecución del trigger [Facturas], porque la eliminación de los elementos de la lista es consecutiva a una llamada a **DELETE SELECTION** desde el trigger de [Facturas].*

Imagine que todas las tablas en este ejemplo tienen triggers activados para todos los eventos de la base de datos. La cascada de triggers será:

```
El trigger de[Facturas] se llama porque el proceso llamante borra una factura
El trigger de[Clientes] se llama porque el trigger de[Facturas] trigger actualiza el campo Ventas_Brutas
El trigger de[Linea_Factura] se llama porque el trigger[Facturas] borra una línea(repetida)
El trigger de[Productos] se llama porque el trigger de [Linea_Factura] actualiza el campo Cantidad_Vendida
El trigger de[Pagos] se llama porque el trigger de [Facturas] borra un pago(repetido)
```

En esta cascada, el trigger de [Facturas] se ejecuta en el nivel 1, los triggers de [Clientes], [Linea_Factura], y [Pagos] en el nivel 2 y el trigger de [Productos] en el nivel 3.

*Desde dentro de los triggers, puede utilizar el comando **Trigger level** para detectar el nivel en el cual se ejecuta un trigger. Además, puede utilizar el comando **TRIGGER PROPERTIES** para obtener información sobre los otros niveles.*

Por ejemplo, si borra un registro de [Productos] a nivel del proceso, el trigger de [Productos] se ejecutará en el nivel 1, no en el nivel 3.

*Con **Trigger level** y **TRIGGER PROPERTIES**, puede identificar la causa de una acción. En nuestro ejemplo, una factura se borra al nivel del proceso. Si borramos un registro de [Clientes] a nivel del proceso, el trigger de [Clientes] debe intentar borrar todas las facturas relacionadas con ese cliente. Esto significa que el trigger [Facturas] será llamado como se llamó anteriormente, pero por otra razón. Desde el trigger de [Facturas], puede detectar si se ejecuta en el nivel 1 ó 2. Si se ejecutó en el nivel 2, puede verificar si fue porque se borro el registro de [Clientes]. Si este es el caso, no tiene que preocuparse en actualizar el campo Ventas_Brutas.*

Trigger event

Trigger event -> Resultado

Parámetro	Tipo	Descripción
Resultado	Entero largo	0 = Fuera de todo evento de trigger 1 = Al guardar un nuevo registro 2 = Al guardar un registro existente 3 = Al borrar un registro

Descripción

*El comando **Trigger event** se llama en un trigger y devuelve un valor numérico que indica el tipo del evento de la base, en otras palabras, la razón por la cual se invoca el trigger.*

*4D ofrece las siguientes constantes predefinidas del tema **Eventos trigger**:*

Constante	Tipo	Valor
On Deleting Record Event	Entero largo	3
On Saving Existing Record Event	Entero largo	2
On Saving New Record Event	Entero largo	1

*Dentro de un trigger, si efectúa operaciones de base de datos sobre varios registros, puede encontrar condiciones (generalmente registros bloqueados) que impiden al trigger ejecutar correctamente las operaciones para las cuales es llamado. Un ejemplo de esta situación es la actualización de varios registros en la tabla [Productos] cuando se añade un registro a la tabla [Facturas]. En este punto, debe detener las operaciones de la base y devolver un error de manera que el proceso llamante sepa que la petición no puede llevarse a cabo. Luego el proceso llamante debe poder cancelar, durante la transacción, las operaciones incompletas efectuadas por el trigger. Cuando se produce este tipo de situación, debe saber en el trigger si está en transacción antes de intentar hacer algo. Para hacer esto, utilice el comando **In transaction**.*

*En 4D, no hay límite, a parte de la memoria disponible, para la llamada de triggers en cascada. Para optimizar la ejecución de un trigger, puede escribir el código de sus triggers dependiendo no sólo del evento de la base, sino también del nivel de la llamada cuando los triggers se llaman en cascada. Por ejemplo, durante una eliminación del evento trigger para la tabla [Facturas], puede no efectuar la actualización del campo [Clientes] Ventas brutas si la eliminación del registro de la tabla [Facturas] es parte de la eliminación en cascada de facturas relacionadas con el registro en la tabla [Clientes] que está siendo eliminado. Para hacer esto, utilice los comandos **Trigger level** y **TRIGGER PROPERTIES**.*

Ejemplo

*Utilice el comando **Trigger event** para estructurar sus triggers de esta manera:*

```
  ` Trigger de la tabla [toda tabla]
C_LONGINT($0)
$0:=0  ` Asegurarse de que la petición de la base será concedida
Case of
   :(Database event=On Saving New Record Event)
  ` Ejecutar las acciones apropiadas para guardar un nuevo registro
   :(Database event=On Saving Existing Record Event)
  ` Ejecutar las acciones apropiadas para guardar un registro existente
   :(Database event=On Deleting Record Event)
  ` Ejecutar las acciones apropiadas para la eliminación de un registro
End case
```

Trigger level

Trigger level -> Resultado

Parámetro	Tipo		Descripción
Resultado	Entero largo		Nivel de ejecución del trigger (0 si está fuera del ciclo de ejecución del trigger)

Descripción

*El comando **Trigger level** devuelve el nivel de ejecución del trigger.*
*Para mayor información sobre niveles de ejecución, consulte el tema **Triggers en cascada** en la sección "Triggers".*

TRIGGER PROPERTIES

TRIGGER PROPERTIES (nivelTrigger ; eventoBase ; numTabla ; regNum)

Parámetro	Tipo		Descripción
nivelTrigger	Entero largo	➡	Nivel de ejecución del trigger
eventoBase	Entero largo	⬅	Evento de base de datos
numTabla	Entero largo	⬅	Número de la tabla
regNum	Entero largo	⬅	Número del registro

Descripción

*El comando **TRIGGER PROPERTIES** devuelve la información sobre el nivel de ejecución del trigger que se pasa en nivelTrigger. Puede utilizar **TRIGGER PROPERTIES** junto con **Trigger level** para efectuar diferentes acciones en función de la cascada del trigger. Para mayor información, consulte la descripción de triggers en cascada en la sección **Triggers**.*

Si pasa un nivel de ejecución de trigger inexistente, el comando devuelve 0 (cero) en todos los parámetros.

*La naturaleza del evento de base de datos para el nivel de ejecución del trigger se devuelve en dbEvent. En el tema **Eventos trigger** se ofrecen las siguientes constantes predefinidas:*

Constante	Tipo	Valor
On Deleting Record Event	Entero largo	3
On Saving Existing Record Event	Entero largo	2
On Saving New Record Event	Entero largo	1

El número de tabla y de registro para el registro relacionado por el evento de base de datos para el nivel de ejecución del trigger se devuelven en tablaNum y regNum.

Variables

- *CLEAR VARIABLE*
- *LOAD VARIABLES*
- *SAVE VARIABLES*

CLEAR VARIABLE

CLEAR VARIABLE (variable)

Parámetro	Tipo		Descripción
variable	Variable	➡	Variable a borrar

Descripción

CLEAR VARIABLE reinicia variable en el valor por defecto de su tipo (por ejemplo, cadena vacía para los tipos Alfa y Texto, 0 para las variables numéricas, ningún elemento para un array, etc.). La variable continúa existiendo en memoria.

La variable que se pasa en variable puede ser una variable local, proceso o interproceso.

Nota: no es necesario borrar las variables proceso al terminar un proceso; 4D las borra automáticamente. De forma similar cada variable local se borra automáticamente cuando el método donde se ubica termina su ejecución.

Ejemplo

En un formulario, utilice una lista desplegable llamada asMiListaDesplegable cuyo único propósito es la interfaz del usuario. En otras palabras, utilice este array durante la entrada de datos, pero una vez cierre el formulario, no utilice más este array. Por lo tanto, durante el evento **On Unload**, borre el array:

```
  ` Método de objeto asMiListaDesplegable
Case of
   :(Form event=On Load)
  ` Inicializar el array de una manera u otra
      ARRAY STRING(63;asMiListaDesplegable;...)
  ` ...
   :(Form event=On Unload)
  ` No necesita más el array
      CLEAR VARIABLE(asMiListaDesplegable)
  ` ...
End case
```

LOAD VARIABLES

LOAD VARIABLES (doc ; variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
doc	Cadena	➡	Documento que contiene el o las variables 4D
variable	Variable	⬅	Variables a recibir los valores

Descripción

El comando ***LOAD VARIABLES*** *carga una o varias variables del documento especificado por documento. El documento debe haberse creado utilizando el comando* ***SAVE VARIABLES****.*

Las variables variable, variable2...variableN son creadas; si ya existen, se sobrescriben.

Si pasa una cadena vacía en documento, aparece una caja de diálogo estándar de apertura de archivos, permitiendo al usuario seleccionar el documento a abrir. Si se elige un documento, la variable sistema ***Document*** *contendrá el nombre del documento.*

En bases de datos compiladas, cada variable debe ser del mismo tipo que las cargadas del disco.

Advertencia: *este comando no soporta variables de tipo array. Para variables de tipo array utilice los comandos del tema BLOB.*

Ejemplo

El siguiente ejemplo carga tres variables de un documento llamado PrefsUsuario:

```
LOAD VARIABLES("PrefsUsuario";vsNombre;vlCodigo;vgIconImagen)
```

Variables y conjuntos del sistema

Si las variables se cargan correctamente, la variable sistema ***OK*** *toma el valor 1; de lo contrario toma el valor 0.*

SAVE VARIABLES

SAVE VARIABLES (doc ; variable {; variable2 ; ... ; variableN})

Parámetro	Tipo		Descripción
doc	Cadena	➡	Nombre del documento en el cual guardar las variables
variable	Variable	➡	Variables a guardar

Descripción

El comando **SAVE VARIABLES** guarda una o varias variables en el documento cuyo nombre se pasa en l parámetro documento.

Las variables no deben ser del mismo tipo, pero tienen que ser de tipo Texto, Numérico, Fecha, Hora, Booleano o Imagen.

Si pasa una cadena vacía en documento, aparece una caja de diálogo estándar de guardar archivos; el usuario puede entonces elegir el documento a crear. En este caso, la variable sistema **Document** toma el nombre del documento si se ha creado.

Si las variables se guardan correctamente, la variable OK toma el valor 1. Si no, OK toma el valor 0.

Nota: cuando escribe variables en documentos con **SAVE VARIABLES**, 4D utiliza un formato de datos interno. Puede recuperar las variables únicamente con el comando **LOAD VARIABLES**. No utilice **RECEIVE VARIABLE** o **RECEIVE PACKET** para leer un documento creado por **SAVE VARIABLES**.

Advertencia: este comando no soporta variables de tipo array. Para estas variables utilice los comandos del tema BLOB.

Ejemplo

El siguiente ejemplo guarda tres variables en un archivo llamado PrefsUsuario:

```
SAVE VARIABLES(" PrefsUsuario";vsNombre;vlCodigo;vgIconImagen)
```

Variables y conjuntos del sistema

Si las variables se guardan correctamente, la variable sistema **OK** toma el valor 1; de lo contrario toma el valor 0.

Ventanas

- *Gestión de ventanas*
- *Tipos de ventanas*
- *CLOSE WINDOW*
- *CONVERT COORDINATES*
- *Current form window*
- *DRAG WINDOW*
- *ERASE WINDOW*
- *Find window*
- *Frontmost window*
- *GET WINDOW RECT*
- *Get window title*
- *HIDE TOOL BAR*
- *HIDE WINDOW*
- *MAXIMIZE WINDOW*
- *MINIMIZE WINDOW*
- *Next window*
- *Open form window*
- *Open window*
- *REDRAW WINDOW*
- *RESIZE FORM WINDOW*
- *SET WINDOW RECT*
- *SET WINDOW TITLE*
- *SHOW TOOL BAR*
- *SHOW WINDOW*
- *Tool bar height*
- *Window kind*
- *WINDOW LIST*
- *Window process*
- *_o_Open external window*
- *Tipos de ventanas (Compatibilidad)*

Gestión de ventanas

Las ventanas se utilizan para mostrar información al usuario. Tienen tres usos principales: la entrada de datos, la visualización de datos, y la visualización de mensajes para el usuario.

Siempre hay por lo menos una ventana abierta. Si es necesario, se añaden barras de desplazamiento, con el fin de permitir al usuario desplazarse en un formulario que es más largo que la ventana. En el entorno Diseño, esta ventana muestra la lista de registros (formulario de salida) o la pantalla de entrada de datos (formulario de entrada). En el entorno Aplicación, esta ventana muestra una pantalla con un gráfico personalizado.

Cuando selecciona un comando de menú en modo Aplicación, la pantalla de bienvenida puede ser reemplazada por datos cuando llama los comandos que muestran formularios. Una vez termina la ejecución de los comandos, la pantalla de bienvenida aparece de nuevo por defecto.

WinRef

*Puede abrir varios tipos de ventanas personalizadas con los comandos **Open window** u **Open form window** (ver la sección **Tipos de ventanas (Compatibilidad)**). Todas las ventanas abiertas por estos comandos son referenciadas con la expresión **refVen**. Un refVen es la identificación única de cada ventana abierta. Es una expresión de tipo Entero largo. Todos los comandos que trabajan con ventanas personalizadas esperan un parámetro refVen.*

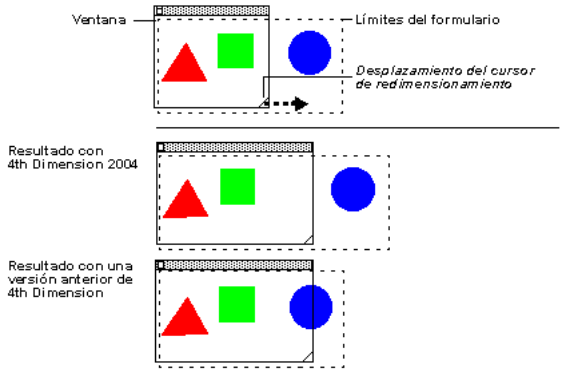
*Cuando no necesite más una ventana personalizada, debe cerrarla utilizando el comando **CLOSE WINDOW** o haciendo clic en la casilla del menú Control (Windows) o Cerrar (Macintosh), si existe.*

*Algunos comandos abren sus propias ventanas, comandos tales como **QR REPORT** y **PRINT LABEL** abren una ventana que se vuelve la ventana del primer plano.*

Si inicia un nuevo proceso y no abre una ventana al comienzo del método de proceso, 4D la creará automáticamente con el tipo por defecto, tan pronto como se muestre un formulario.

Separadores pushers

El borde derecho e inferior son por defecto separadores "pusher". Esto significa que los objetos que se encuentran a la derecha o debajo de los límites de una ventana se corren automáticamente a la derecha o hacia abajo si se agranda la ventana:



*Este mecanismo le permite administrar ventanas retractables como la del Explorador Window (ver el ejemplo del comando **FORM SET SIZE**).*

***Nota:** este principio no funciona con ventanas que tengan barras de desplazamiento.*

Coordenadas de las ventanas y modo "derecha a izquierda"

En los comandos de gestión de ventanas, las coordenadas de las ventanas son determinadas con respecto a un punto de origen generalmente ubicado en la parte superior izquierda de la ventana/pantalla.
Sin embargo, cuando el modo "derecha a izquierda" está activado para la aplicación, las coordenadas se invierten y el punto de origen pasa a la parte superior derecha de ventana/pantalla. Por lo tanto, en este modo las coordenadas horizontales utilizadas por los siguientes comandos deben invertirse:

Open window
Open form window
_o_Open external window
GET WINDOW RECT
SET WINDOW RECT
Find window

***Nota:** para mayor información sobre el modo "derecha a izquierda", consulte el Manual de Diseño y la descripción del comando **SET DATABASE PARAMETER**.*

Tipos de ventanas

Presentación

*Puede utilizar una de las siguientes constantes predefinidas (tema "**Abrir ventana formulario**" para especificar el tipo de ventana a abrir con **Open form window**:*

Constante	Tipo	Valor
Modal form dialog box	Entero largo	1
Movable form dialog box	Entero largo	5
Plain form window	Entero largo	8
Pop up form window	Entero largo	32
Sheet form window	Entero largo	33
Toolbar form window	Entero largo	35
Palette form window	Entero largo	1984
Form has no menu bar	Entero largo	2048
Form has full screen mode Mac	Entero largo	65536
Controller form window	Entero largo	133056

Esta sección muestra cada tipo de ventana en Windows (izquierda) y macOS (derecha).

Ventanas modales

Una ventana modal coloca al usuario en un estado (o "modo") donde sólo puede actuar dentro de esta ventana. Mientras se muestre la ventana modal, los comandos de menú y las otras ventanas de la aplicación son inaccesibles. Para cerrar una ventana modal, el usuario debe validarla, cancelarla, o elegir una de las opciones que ofrece. Las cajas de diálogo de alerta son ejemplos típicos de ventanas modales.

En 4D, las ventanas de tipo 1 y 5 son ventanas modales.

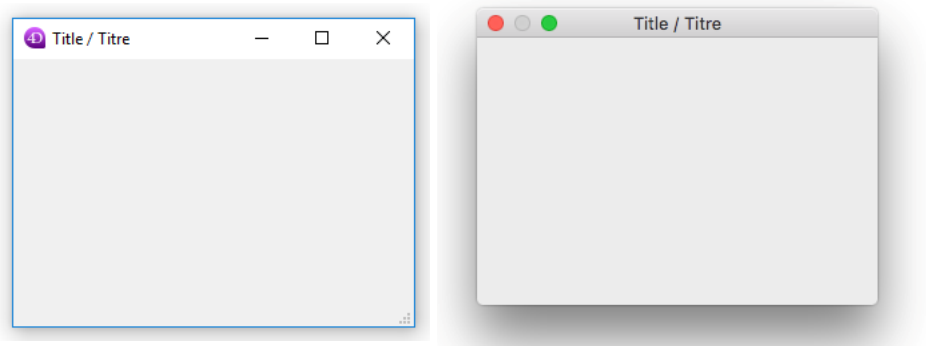
***Nota:** una ventana modal siempre permanece en el primer plano. Por consiguiente, cuando una ventana modal llama a una ventana no modal, esta última ventana se muestra al fondo, incluso si se llamó después de la ventana modal. Por lo tanto evite este tipo de operación.*

Por el contrario, cuando una ventana modal llama a otra ventana modal, la última ventana se mostrará en el primer plano.

Diálogo modal (1)

- *Puede tener un título: no*
- *Puede tener una casilla Cerrar o un equivalente: no*
- *Puede redimensionarse: no*
- *Puede minimizarse/maximizarse o hacerle zoom: no*
- *Adaptada a las barras de desplazamiento: no*
- *Uso: **DIALOG**, **ADD RECORD** o equivalente*
- *Las ventanas de este tipo son modales*

Diálogo modal desplegable (5)

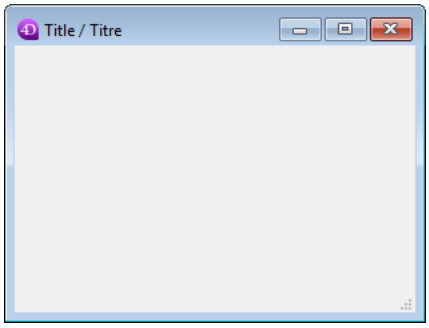


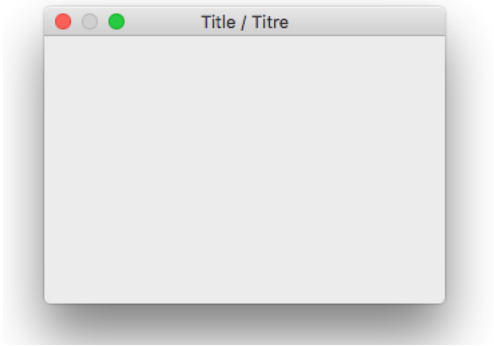
- *Puede tener un título: sí*

- *Puede tener una casilla cerrar o un equivalente: sí*
- *Puede redimensionarse: sí (*)*
- *Puede minimizarse/maximizarse o hacerle zoom: sí (*)*
- *Adaptada a las barras de desplazamiento: no*
- *Uso: **DIALOG**, **ADD RECORD**(...;...;*) o equivalente*
- *Las ventanas de este tipo son modales, pero pueden moverse.*

(*) No disponible en las versiones 32 bits de 4D en Windows.

Ventana estándar (8)





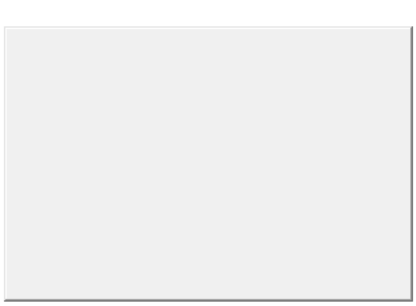
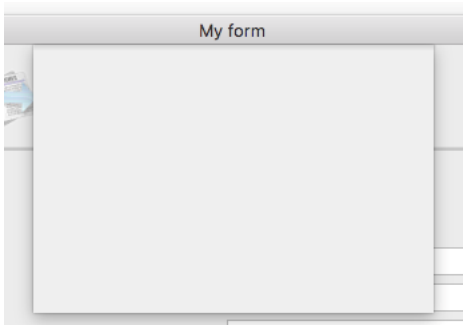
- *Puede tener un título: sí*
- *Puede tener una casilla Cerrar o un equivalente: sí*
- *Puede redimensionarse: sí*
- *Puede minimizarse/maximizarse o hacerle zoom: sí*
- *Adaptada a las barras de desplazamiento: sí*
- *Uso: entrada de datos con barras de desplazamiento, **DISPLAY SELECTION**, **MODIFY SELECTION**, etc.*

Ventana Pop up (32)

Este tipo de ventana tiene las siguientes características avanzadas específicas:

- *La ventana no puede tener una caja cerrar pero se cierra automáticamente y el evento "cancelar" se pasa a la ventana cuando:*
  - ocurre un clic fuera de la ventana;
  - la ventana de fondo o la ventana MDI (Multiple Document Interface) se desplaza;
  *- el usuario hace clic en la tecla **Esc**.*
- *Esta ventana se muestra delante de su ventana "padre" (no debe utilizarse como ventana principal del proceso). La ventana de fondo no está desactivada. Sin embargo, no recibe más eventos.*
- *No es posible redimensionar o desplazar la ventana utilizando el ratón; sin embargo, al realizar estas acciones por programación, se optimiza el redimensionamiento de los elementos del fondo.*
- *Uso: este tipo de ventana se adapta particularmente al manejo de los menús pop-up asociados a los botones 3D de tipo "bevel" o "barras de herramientas".*
- *Limitaciones:*
  - *No es posible mostrar objetos de menú pop-up dentro de este tipo de ventana.*
  - *A partir de la versión 13 de 4D, este tipo de ventana no permite mostrar mensajes de ayuda en Mac OS.*

Ventana hoja (33)

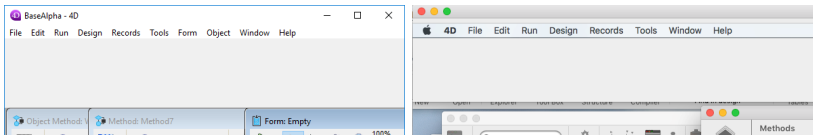



Las ventanas hoja (sheet windows) son específicas para Mac OS. Estas ventanas "descienden" de la barra de título de la ventana principal utilizando una animación y se muestran sobre la ventana principal. Se centran automáticamente en la ventana principal.

Sus propiedades son idénticas a las de las cajas de diálogo modales. Por lo general se utilizan para realizar una acción directamente relacionada con la que se lleva a cabo en la ventana principal.

- *Puede crear una ventana hoja únicamente bajo Mac OS si la última ventana abierta es visible y de tipo documento (form).*
- *El comando abre una ventana de tipo 1 (diálogo modal) en lugar de una de tipo 33:*
  - si la última ventana abierta no es visible o no es de tipo documento,
  - bajo Windows.
- *Como una ventana hoja debe dibujarse sobre un formulario, su visualización se rechaza en el evento On Load del primer formulario cargado en la ventana (ver el ejemplo 3 del comando **Open form window**).*
- *Uso: **DIALOG**, **ADD RECORD**(...;...*) o equivalente, bajo Mac OS (no estándar bajo Windows).*

Ventana barra herramientas (35)

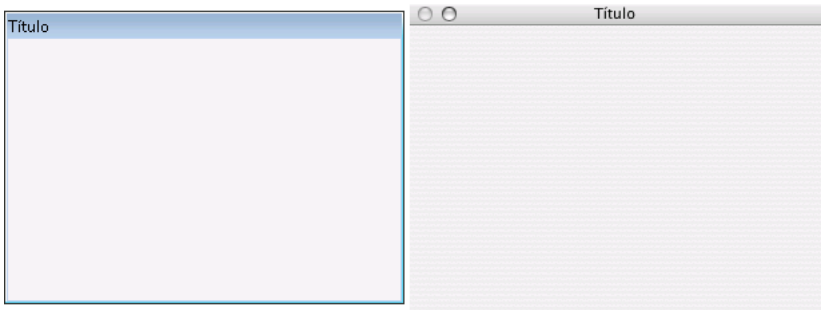


Una ventana barra de herramientas se crea con la ubicación, tamaño y propiedades gráficas de la barra de herramientas, es decir:

- *La ventana se mostrará siempre justo debajo de la barra de menús.*
- *El tamaño horizontal de la ventana se ajustará automáticamente para llenar todo el espacio horizontal disponible en el escritorio (en macOS y en Windows en modo SDI) o dentro de la ventana de la aplicación 4D (en Windows en modo MDI). El tamaño vertical de la ventana se basa en las propiedades del formulario, como todos los otros tipos de ventanas formulario.*
- *La ventana no tiene borde, no se puede mover y no se puede cambiar el tamaño de forma manual.*
- *No es posible crear dos ventanas de la barra de herramientas diferentes al mismo tiempo en el mismo proceso (de lo contrario se genera el error -10613 "No se puede crear dos ventanas formulario de tipo barra de herramientas").*

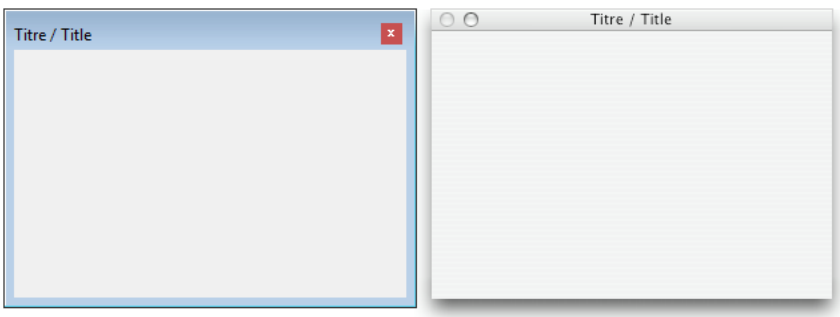
***Barra de herramientas y modo pantalla completa en macOS:** si su aplicación muestra a la vez una ventana barra de herramientas y una ventana estándar que admiten el modo de pantalla completa (opción (Form has full screen mode Mac), las normas de interfaz requieren que se oculte la barra de herramientas cuando una ventana estándar pase a modo pantalla completa. Para saber si una ventana ha cambiado al modo pantalla completa, simplemente pruebe si su tamaño vertical es exactamente el mismo que la altura de la pantalla (consulte el comando **HIDE TOOL BAR**).*

Ventana Paleta (1984)



Este tipo de ventana le permite generar ventanas flotantes. La característica principal de las ventanas flotantes es que permanecen en el primer plano incluso si el usuario hace clic en otra ventana del proceso. Las ventanas flotantes se utilizan generalmente para mostrar información permanente o barras de herramientas.

Ventana Controller form (133056)



Este tipo de ventana es similar a Palette form window con la siguiente especificidad: en Windows, la ventana flotante será referenciada por un icono en la barra de tareas (en Windows, las paletas flotantes regulares no se muestran en la barra de tareas).

*Este tipo de ventana es útil cuando la base de datos se ejecuta en **Modo SDI en Windows** en Windows. En este modo, las paletas flotantes se ocultan cuando su aplicación principal se mueve al fondo. Por lo tanto, si su interfaz de base de datos se basa en una única ventana flotante (por ejemplo, para mostrar una vista de monitor), debe utilizar un Controller form window para referenciar la aplicación en la barra de tareas y asegúrarse de que permanecerá accesible incluso si se ha movido al fondo.*

Por ejemplo

```
$win:=Open form window("myMonitor";Controller form window;On the left;Vertically centered)
```

Nota: *bajo macOS, este tipo de ventana se comporta como una Palette form window regular.*

Sin barra de menús (2048)

Esta opción está destinada a ser utilizada cuando la base de datos se ejecuta en ***Modo SDI en Windows****.*
En este contexto, todas las ventanas de su aplicación muestran de forma predeterminada la barra de menús del proceso actual. Si desea abrir una ventana sin barra de menús, debe agregar la constante Form has no menu bar al parámetro tipo. Por ejemplo, este código crea una ventana de formulario plana sin barra de menú en una aplicación SDI en Windows:

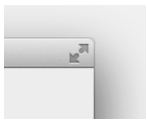
```
$win:=Open form window("myPanel";Plain form window+Form has no menu bar;Horizontally centered;At the top)
```

Nota: *esta opción no tiene efecto:*

- *En una aplicación macOS,*
- *En una aplicación Windows en modo MDI.*

Con modo pantalla completa Mac (65536)

La opción de "pantalla completa" está disponible a partir de 4D v14 en OS X para las ventanas de tipo de documento. Cuando se utiliza esta opción, el botón "Pantalla completa" se muestra en la esquina superior derecha de la ventana:



Cuando el usuario hace clic en este icono, la ventana pasa a pantalla completa y 4D oculta automáticamente la barra de herramientas principal.
Para utilizar esta opción, añada la constante Form has full screen mode Mac al parámetro tipo. Por ejemplo, este código crea una ventana formulario con botón pantalla completa bajo macOS:

```
$win:=Open form window([Interface];"User_Choice";Plain form window+Form has full screen mode Mac)
DIALOG([Interface];"User_Choice")
```

Nota: *bajo Windows, esta opción no tiene efecto.*

CLOSE WINDOW

CLOSE WINDOW {(ventana)}

Parámetro	Tipo		Descripción
ventana	WinRef	➡	Número de referencia, o Ventana del primer plano del proceso, si este parámetro se omite

Descripción

CLOSE WINDOW cierra la ventana activa abierta por el comando **Open window** u **Open form window** en el proceso actual. **CLOSE WINDOW** no hace nada si no hay una ventana personalizada abierta; no cierra las ventanas sistema. **CLOSE WINDOW** tampoco tiene efecto si se llama mientras que un formulario esté activo en la ventana. Debe llamar **CLOSE WINDOW** cuando haya terminado de utilizar una ventana abierta por **Open window** u **Open form window.** Es inútil pasar un número a **CLOSE WINDOW** cuando lo utiliza para cerrar las ventanas abiertas por **Open window** u **Open form window,** ya que **CLOSE WINDOW** siempre cerrará la última ventana creada por uno de estos comandos. Si pasa un número de referencia de ventana externa en el parámetro extWindowRef, **CLOSE WINDOW** cierra la ventana externa especificada. Para mayor información sobre ventanas externas, consulte la función **Open external window**.

Ejemplo

El siguiente ejemplo abre una ventana formulario y crea nuevos registros con el comando **ADD RECORD**. Una vez añadidos los registros, la ventana se cierra con **CLOSE WINDOW**:

```
FORM SET INPUT([Employees];"Entry")
$winRef:=Open form window([Employees];"Entry")
Repeat
   ADD RECORD([Employees]) //Añadir un nuevo registro de empleado
Until(OK=0) //Bucle hasta que el usuario cancela
CLOSE WINDOW //Cierre de la ventana
```

CONVERT COORDINATES

CONVERT COORDINATES (coordX ; coordY ; de ; a)

Parámetro	Tipo		Descripción
coordX	Variable entero largo	➡	Coordenada horizontal de un punto (inicial)
		⬅	Coordenada horizontal de un punto (convertido)
coordY	Variable entero largo	➡	Coordenada vertical de un punto(inicial)
		⬅	Coordenada vertical de un punto (convertido)
de	Entero largo	➡	Sistema de coordenadas de origen
a	Entero largo	➡	Sistema de coordenadas a convertir el punto

Descripción

*El comando **CONVERT COORDINATES**convierte las coordenadas (x,y) de un punto de un sistema de coordenadas a otro. Los sistemas de coordenadas de entrada y salida soportados son los formularios (y subformularios), las ventanas y la pantalla. Por ejemplo, puede utilizar este comando para obtener las coordenadas en el formulario principal de un objeto perteneciente a un subformulario. Este principio facilita la creación de menús contextuales en cualquier posición personalizada.*

En coordX y coordY, pase las variables que contienen las coordenadas (x,y) del punto que desea convertir. Después de ejecutar el comando, estas variables contendrán los valores convertidos.

*En el parámetro de, pase el sistema de coordenadas inicial del punto de entrada y en el parámetro a, pase el sistema de coordenadas al que se debe convertir. Ambos parámetros pueden tomar el valor de una de las siguientes constantes, añadidas al tema "**Ventana**":*

Constante	Tipo	Valor	Comentario
XY Current form	Entero largo	1	El origen es la esquina superior izquierda del formulario actual
XY Current window	Entero largo	2	El origen es la esquina superior izquierda de la ventana actual
XY Main window	Entero largo	4	En Windows: origen es la esquina superior izquierda de la ventana principal; en OS X: igual que XY Screen
XY Screen	Entero largo	3	El origen es la esquina superior izquierda de la pantalla principal (igual que para el comando **SCREEN COORDINATES**)

Cuando este comando se llama desde el método de un subformulario o un objeto de un subformulario, y si uno de los selectores es XY Current form, a continuación, las coordenadas son relativas al subformulario en sí, no a su formulario padre.

*Al convertir desde/hacia la posición de una ventana de formulario (por ejemplo, una conversión desde los resultados de **GET WINDOW RECT**, o con los valores pasados a **Open form window**), XY Main window debe ser utilizado, ya que es el sistema de coordenadas utilizado por los comandos de la ventana en Windows. También se puede utilizar para este propósito en OS X, donde es equivalente a XY Screen.*

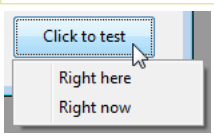
Cuando de es XY Current form y el punto se encuentra en la sección del cuerpo de un formulario listado, el resultado depende del contexto de llamada del comando:

- *Si el comando se llama en el evento On Display Detail, el punto resultante se encuentra en el perímetro de visualización del registro mostrado en la pantalla.*
- *Si el comando se llama fuera de un evento On Display Detail pero mientras se está editando un registro, el punto resultante se encuentra en el perímetro de visualización del registro que se está editando.*
- *De lo contrario, el punto resultante se encuentra en la pantalla del primer registro.*

Ejemplo 1

Usted quiere abrir un menú emergente situado en la esquina inferior izquierda del objeto "MyObject".

```
// OBJECT GET COORDINATES trabaja en el sistema de coordenadas del formulario actual
// El menú emergente dinámico utiliza el sistema de coordenadas de la ventana actual
// Tenemos que convertir los valores
C_LONGINT($left;$top;$right;$bottom)
C_TEXT($menu)
OBJECT GET COORDINATES(*;"MyObject";$left;$top;$right;$bottom)
CONVERT COORDINATES($left;$bottom;XY Current form;XY Current window)
$menu:=Create menu
APPEND MENU ITEM($menu;"Right here")
APPEND MENU ITEM($menu;"Right now")
Dynamic pop up menu($menu;"";$left;$bottom)
RELEASE MENU($menu)
```



Ejemplo 2

Usted quiere abrir una ventana emergente en la posición del cursor del ratón. En Windows, es necesario convertir las coordenadas desde ***GET MOUSE*** *(con el parámetro *) devolviendo valores basados en la posición de la ventana MDI:*

```
C_LONGINT($mouseX;$mouseY;$mouseButtons)
C_LONGINT($window)
GET MOUSE($mouseX;$mouseY;$mouseButtons)
CONVERT COORDINATES($mouseX;$mouseY;XY Current window;XY Main window)
$window:=Open form window("PopupWindowForm";Pop up form window;$mouseX;$mouseY)
DIALOG("PopupWindowForm")
CLOSE WINDOW($window)
```

Current form window

Current form window -> Resultado

Parámetro	*Tipo*	*Descripción*
Resultado	*WinRef*	*Número de referencia de la ventana del formulario actual*

Descripción

*El comando **Current form window** devuelve la referencia de la ventana del formulario actual. Si ninguna ventana ha sido definida para el formulario actual, el comando devuelve 0.*

*La ventana del formulario actual puede ser generada automáticamente por un comando tal como **ADD RECORD**, después de una acción de usuario o utilizando los comandos **Open window** u **Open form window**.*

DRAG WINDOW

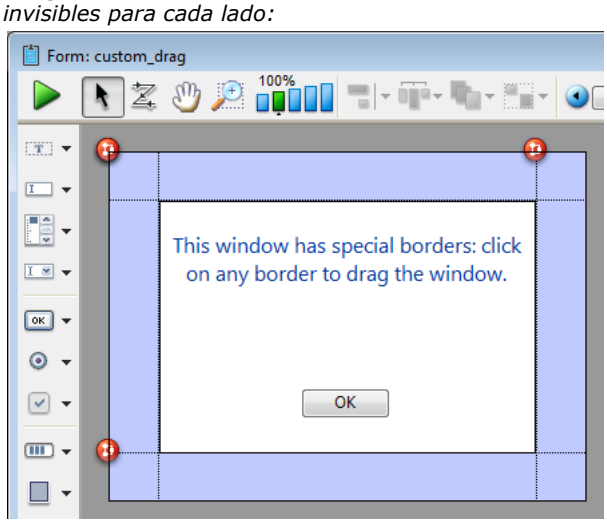
DRAG WINDOW
Este comando no requiere parámetros

Descripción

*El comando **DRAG WINDOW** permite arrastrar la ventana en la cual el usuario hace clic para desplazarla en función de los movimientos del ratón. Generalmente este comando se llama desde un método de objeto de un objeto que pueda responder instantáneamente a los clics del ratón (por ejemplo un botón invisible).*

Ejemplo

El siguiente formulario, mostrado en el editor de formularios, contiene un fondo de color, sobre el cual hay cuatro botones invisibles para cada lado:



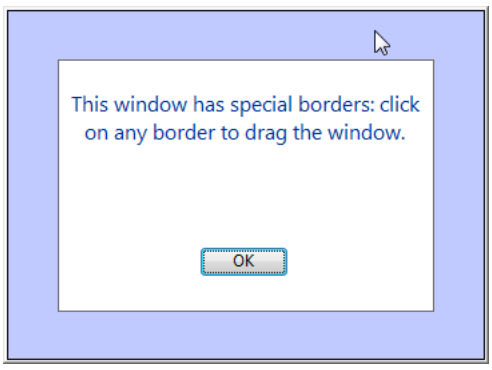
Cada botón está asociado al siguiente método:

```
DRAG WINDOW  //Comenzar a arrastrar la ventana al hacer clic
```

Después de la ejecución del siguiente método proyecto:

```
$winRef:=Open form window("custom_drag";Modal form dialog box)
DIALOG("custom_drag")
CLOSE WINDOW
```

Obtiene una ventana similar a esta:



Luego puede arrastrar la ventana haciendo clic en cualquiera de los bordes.

ERASE WINDOW

ERASE WINDOW {(ventana)}

Parámetro	Tipo		Descripción
ventana	WinRef	➡	Número de referencia de ventana o Ventana del primer plano del proceso actual, si se omite

Descripción

*El comando **ERASE WINDOW** borra el contenido de la ventana cuyo número de referencia se pasa en ventana.*

*Si omite el parámetro ventana, **ERASE WINDOW** borra el contenido de la ventana del primer plano del proceso actual.*

*Generalmente, utilizará **ERASE WINDOW** en combinación con **MESSAGE** y **GOTO XY**. En este caso, **ERASE WINDOW** borra el contenido de la ventana y coloca el cursor en la esquina superior izquierda de la ventana, es decir a la posición correspondiente a **GOTO XY** (0; 0).*

*No confunda **ERASE WINDOW**, que borra el contenido de una ventana, con **CLOSE WINDOW**, que suprime la ventana de la pantalla.*

Find window

Find window (izquierda ; superior {; parteVentana}) -> Resultado

Parámetro	Tipo		Descripción
izquierda	Entero largo	➡	Coordenada global izquierda
superior	Entero largo	➡	Coordenada global superior
parteVentana	Entero largo	⬅	Número de parte de ventana
Resultado	WinRef	➲	Número de referencia de la ventana

Descripción

*El comando **Find window**devuelve (si existe) el número de referencia de la primera ventana "tocada" por el punto cuyas coordenadas se pasan en izquierda y superior.*

Las coordenadas deben ser expresadas con relación a la esquina superior izquierda del área de contenido de la ventana de aplicación (Windows) o de la pantalla principal (Macintosh).

*El parámetro parteVentana devuelve 3 si se toca la ventana, de lo contrario 0. (**Nota de compatibilidad:** a partir de 4D v14, las constantes del tema **Buscar ventana** son obsoletas).*

Frontmost window

Frontmost window {(*)} -> Resultado

Parámetro	Tipo		Descripción
*	Operador	⇒	Si se especifica, tiene en cuenta las ventanas flotantes Si se omite, ignora las ventanas flotantes
Resultado	WinRef	⇐	Número de referencia de ventana

Descripción

*El comando **Frontmost window** devuelve el número de referencia de la ventana ubicada en el primer plano.*

GET WINDOW RECT

GET WINDOW RECT (izquierda ; superior ; derecha ; inferior {; ventana})

Parámetro	Tipo		Descripción
izquierda	Entero largo	⇐	Coordenada izquierda del interior de la ventana
superior	Entero largo	⇐	Coordenada superior del interior de la ventana
derecha	Entero largo	⇐	Coordenada derecha del interior de la ventana
inferior	Entero largo	⇐	Coordenada inferior del interior de la ventana
ventana	WinRef	⇒	Número de referencia de la ventana o Ventana del primer plano del proceso si se omite o Ventana MDI si -1 (Windows)

Descripción

*El comando **GET WINDOW RECT** devuelve las coordenadas globales de la ventana cuyo número de referencia se pasa en ventana. Si la ventana no existe, las variables de los parámetros no cambian.*

*Si omite el parámetro ventana, **GET WINDOW RECT** se aplica a la ventana del primer plano del proceso actual.*

Las coordenadas devueltas se expresan con relación a la esquina superior izquierda del área de contenido de la ventana de aplicación (modo Windows MDI) o de la pantalla principal (macOS y modo Windows SDI). Las coordenadas devuelven el rectángulo correspondiente al área de contenido de la ventana (excluyendo las barras de títulos y los bordes).

***Nota:** bajo Windows, si pasa -1 en ventana, **GET WINDOW RECT** devuelve las coordenadas de la ventana de aplicación (ventana MDI). Estas coordenadas corresponden al área de contenido de la ventana (excluyendo barras de menús y bordes). En este caso en modo SDI, **GET WINDOW RECT** devuelve (0;0;0;0) como coordenadas.*

Ejemplo

*Ver el ejemplo del comando **WINDOW LIST**.*

Get window title

Get window title {(ventana)} -> Resultado

Parámetro	Tipo		Descripción
ventana	WinRef	→	Número de referencia de la ventana o Ventana del primer plano del proceso actual si se omite
Resultado	Cadena	↩	Título de la ventana

Descripción

*El comando **Get window title** devuelve el título de la ventana cuyo número de referencia se pasa en ventana. Si la ventana no existe, se devuelve una cadena vacía.*

*Si omite el parámetro ventana, **Get window title** devuelve el título de la ventana del primer plano del proceso actual.*

Ejemplo

*Ver ejemplo del comando **SET WINDOW TITLE**.*

HIDE TOOL BAR

HIDE TOOL BAR
Este comando no requiere parámetros

Descripción

*El comando **HIDE TOOL BAR** maneja la visualización de las barras de herramientas personalizadas creadas por el comando **Open form window** para el proceso actual.*
*Si una ventana barra de herramientas ha sido creado por el comando **Open form window** con la opción Toolbar form window, el comando oculta esta ventana. Si la ventana barra de herramientas ya estaba oculta o si ninguna ventana de este tipo ha sido creada, el comando no hace nada.*

Ejemplo

En OS X, se ha definido una barra de herramientas personalizada y una ventana estándar que tiene la opción Has full screen mode Mac. Cuando una ventana estándar es maximizada por un usuario mientras se muestra la ventana de la barra de herramientas, usted no desea que la barra de herramientas solape la ventana maximizada.
*Para evitar esto, en el evento formulario On Resize de la ventana estándar, es necesario detectar cuando la ventana pasa a modo pantalla completa y luego llamar **HIDE TOOL BAR**:*

```
Case of
   :(Form event=On Resize)
      GET WINDOW RECT($left;$top;$right;$bottom)
      If(Screen height=($bottom-$top))
         HIDE TOOL BAR
      Else
         SHOW TOOL BAR
      End if
End case
```

HIDE WINDOW

HIDE WINDOW {(ventana)}

Parámetro	Tipo		Descripción
ventana	WinRef	➡	Número de referencia de la ventana o Ventana del primer plano del proceso actual, si se omite

Descripción

*El comando **HIDE WINDOW** permite ocultar la ventana cuyo número de referencia se pasa en ventana o, si se omite este parámetro, la ventana del primer plano del proceso actual. Este comando permite, por ejemplo, en un proceso con varias ventanas, mostrar únicamente la ventana activa.*

La ventana desaparece de la pantalla pero permanece abierta. Puede aplicar por programación cambios soportados por las ventanas 4D.

*Para mostrar una ventana oculta por el comando **HIDE WINDOW**:*

- *Utilice el comando **SHOW WINDOW** y pase el número de referencia de la ventana.*
- *Utilice la página **Proceso** del Explorador de ejecución. Seleccione el proceso en el cual se manipula la ventana, luego haga clic en el botón **Mostrar**.*

*Para ocultar todas las ventanas de un proceso, utilice el comando **HIDE PROCESS**.*

Ejemplo

Este ejemplo corresponde a un método de un botón ubicado en un formulario de entrada. Este botón abre una caja de diálogo en una nueva ventana del mismo proceso. En este ejemplo, el usuario quiere ocultar las otras ventanas del proceso (un formulario de entrada y una paleta de herramientas) mientras muestra la caja de diálogo. Una vez validada la caja de diálogo, otras ventanas de proceso se muestran nuevamente.

```
` Método de objeto del botón "Informacion"

HIDE WINDOW(Entrada) ` Ocultar la ventana de entrada
HIDE WINDOW(Paleta) ` Ocultar la paleta
$Infos:=Open window(20;100;500;400;8) ` Crear la ventana de información
... ` Colocar aquí las instrucciones necesarias para la administración del diálogo
CLOSE WINDOW($Infos) ` Cerrar el diálogo
SHOW WINDOW(Entrada)
SHOW WINDOW(Paleta) ` Mostrar las otras ventanas del proceso
```

MAXIMIZE WINDOW

MAXIMIZE WINDOW {(ventana)}

Parámetro	Tipo		Descripción
ventana	WinRef	➡	Número de referencia de la ventana Si se omite = todas las ventanas (Windows) o Ventana del primer plano del proceso actual (Mac OS)

Descripción

El comando ***MAXIMIZE WINDOW*** *provoca el zoom de la ventana cuyo número de referencia se pasa en ventana. Si se omite este parámetro, el efecto es el mismo pero se aplica a todas las ventanas de la aplicación (Windows) o de la ventana del primer plano del proceso actual (Mac OS).*

Este comando tiene el mismo efecto que un clic en el zoom de una ventana de la aplicación 4D. En Windows, la ventana que desea maximizar debe tener un cuadro de zoom. Si el tipo de ventana no tiene un cuadro de zoom, el comando no hace nada.

Un clic posterior en la cuadro de zoom o la llamada al comando ***MINIMIZE WINDOW*** *reduce la ventana a su tamaño inicial. En Windows, una llamada a* ***MINIMIZE WINDOW*** *sin parámetros hace que todas las ventanas de la aplicación vuelvan a su tamaño inicial.*

Si ventana ya está maximizada, el comando no hace nada.

En Windows

El tamaño de la ventana se incrementa para coincidir con el tamaño actual de la ventana de la aplicación (modo MDI) o la pantalla (modo SDI). La ventana maximizada pasa al primer plano. Si no pasa el parámetro ventana, el comando se aplica a todas las ventanas de la aplicación.

Zoom (botón para agrandar) bajo Windows

En casos en que se aplica el comando a una ventana cuyo tamaño está sujeto a restricciones (por ejemplo, una ventana formulario):

- *Si ninguna restricción de tamaño está en conflicto con el tamaño objetivo, la ventana se "maximiza" (es decir, se restaura del tamaño de la ventana MDI ("Multiple Document Interface"); su barra de título y sus bordes están ocultos y su botones de control, minimizar, restaurar y cerrar, se ubican a la derecha de la barra de menú de la aplicación).*
- *Si al menos una restricción de tamaño está en conflicto (por ejemplo, si el ancho de la ventana MDI es 100 y el ancho máximo de la ventana de formulario es 80), la ventana no se "maximiza", sólo se restaura a su tamaño máximo permitido. Este tamaño se define ya sea por la ventana MDI, o por la restricción. De esta manera, la interfaz sigue siendo coherente cuando se redimensionan ventanas con restricciones.*

En Mac OS

La ventana se agranda de manera que pueda mostrar la totalidad de su contenido. Si no pasa el parámeto ventana, el comando se aplica a la ventana del primer plano del proceso actual.

Zoom en Mac OS

- *El zoom se basa en el contenido de la ventana; así, el comando debe llamarse en un contexto en el que se definen el contenido de la ventana, por ejemplo, en un método formulario. De lo contrario, el comando no hace nada.*
- *La ventana está dimensionada en su tamaño "máximo". Si la ventana es un formulario cuyo tamaño se define en las Propiedades del formulario, el tamaño de la ventana se establece en esos valores.*

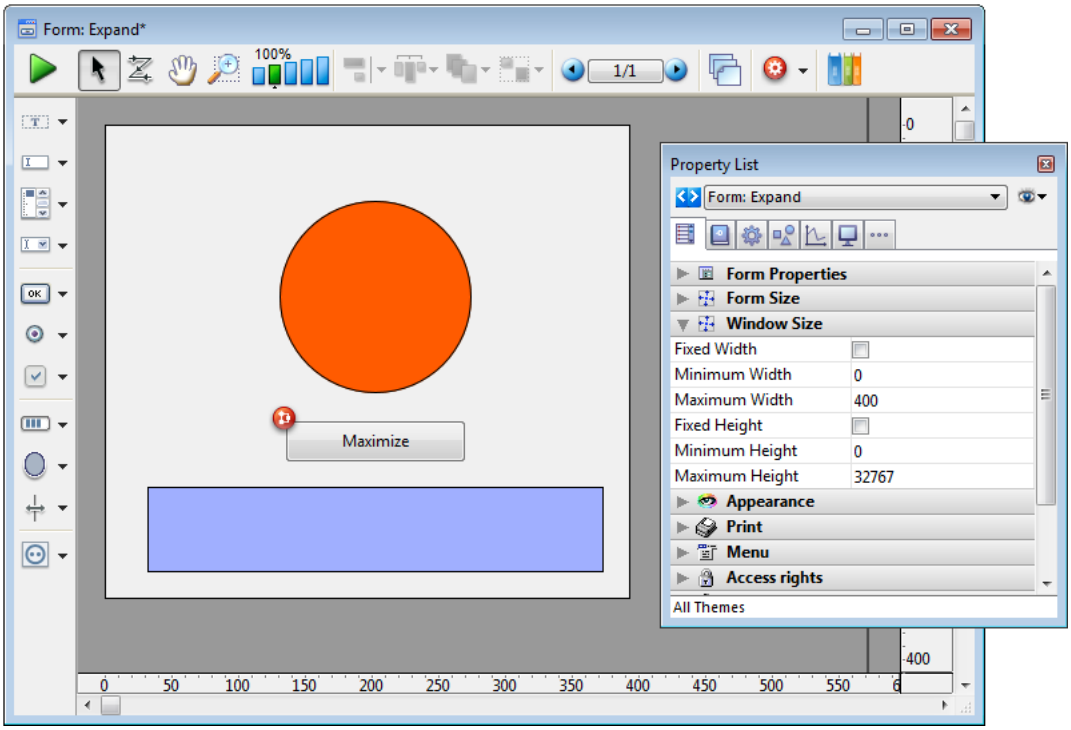
Ejemplo 1

Usted quiere que su formulario se abra en una ventana "abierta totalmente". Para alcanzar esto, usted coloca el siguiente código en el método de formulario:

```
` Método de formulario
MAXIMIZE WINDOW
```

Ejemplo 2

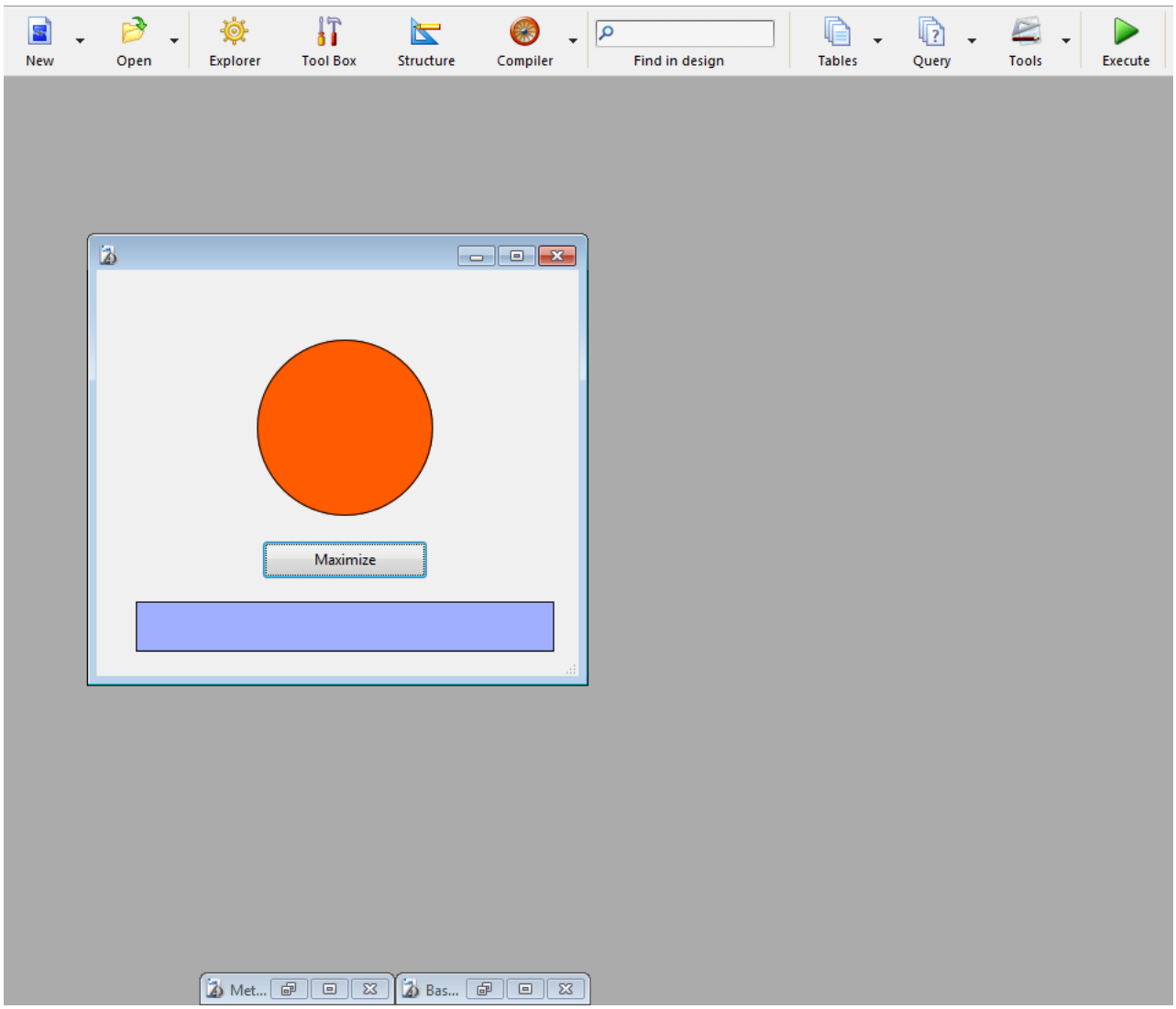
Este ejemplo ilustra cómo las restricciones de tamaño se manejan en Windows (modo MDI). El siguiente formulario tiene una restricción de tamaño (ancho máximo=400):



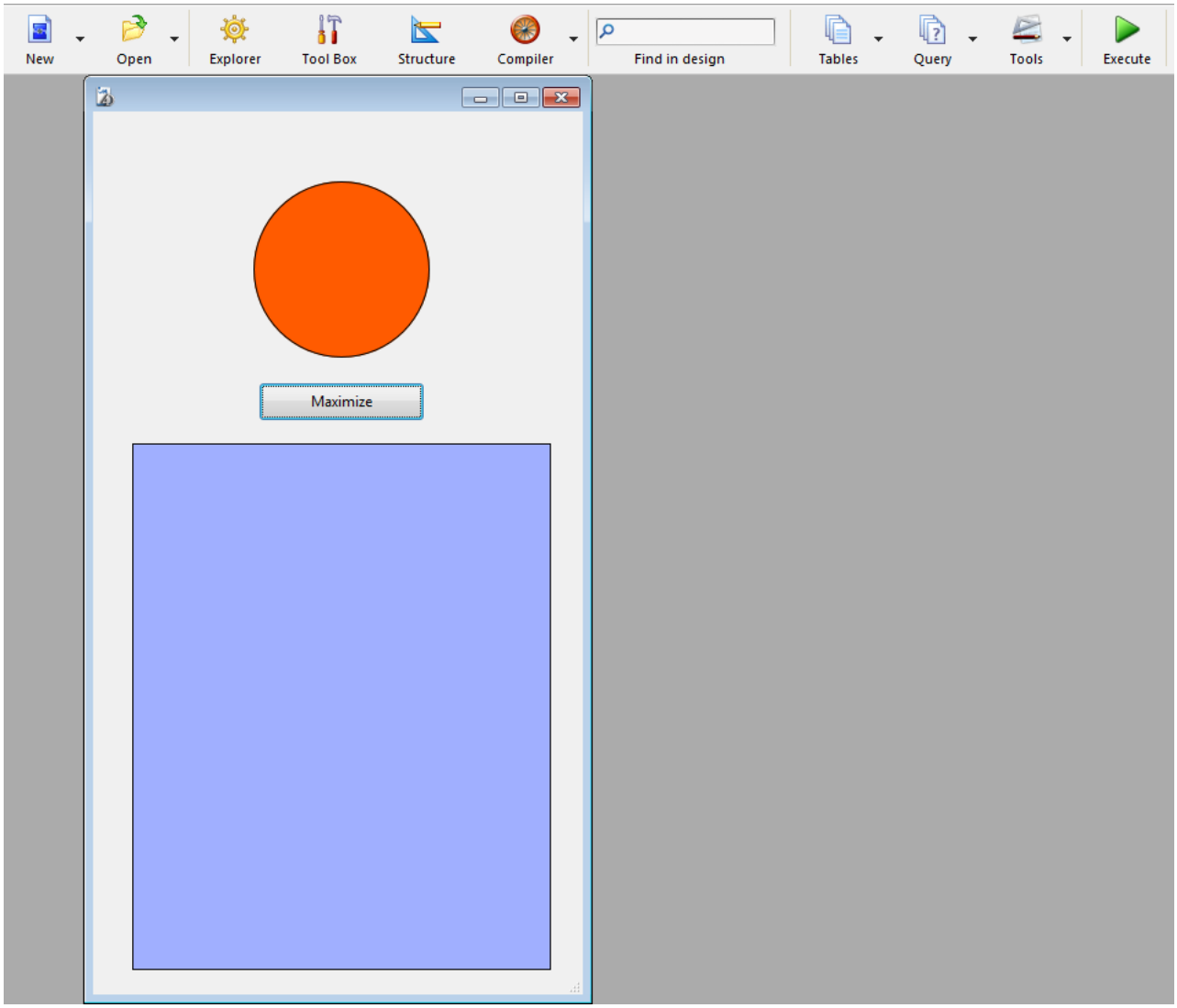
El método del botón contiene únicamente:

```
MAXIMIZE WINDOW (Current form window)
```

En este contexto, si el usuario hace clic en el botón:



... la ventana no se "maximiza"; sólo aumenta el alto:


New
Open
Explorer
Tool Box
Structure
Compiler
Find in design
Tables
Query
Tools
Execute
Maximize

MINIMIZE WINDOW

MINIMIZE WINDOW {(ventana)}

Parámetro	Tipo		Descripción
ventana	WinRef	➡	Número de referencia de la ventana o si se omite todas las ventanas del primer plano del proceso actual (Windows) o ventana del primer plano del proceso actual (Mac OS)

Descripción

*El comando **MINIMIZE WINDOW** define el tamaño de la ventana, cuyo número se pasó en ventana, del tamaño que tenía antes de ser maximizada. Si se omite ventana, el comando aplica a cada ventana de la aplicación (Windows) o a la ventana del primer plano del proceso actual (en Mac OS).*

Este comando tiene el mismo efecto que un clic en la casilla de reducción de la aplicación 4D:

En Windows

La ventana vuelve a su tamaño inicial, es decir, su tamaño antes de ser maximizada. Si se omite el parámetro ventana, todas las ventanas de la aplicación son redimensionadas a su tamaño inicial.

Casilla de reducción en Windows

Bajo Mac OS

El tamaño de la ventana se reduce a su tamaño inicial (su tamaño antes de ser maximizada). Si el parámetro ventana se omite, la ventana del primer plano del proceso actual retorna a su tamaño inicial.

Casilla de zoom/reducción bajo Mac OS

*Si las ventanas a las cuales se aplica este comando no fueron maximizadas previamente (manualmente o utilizando **MAXIMIZE WINDOW**), o si el tipo de ventana no incluye una casilla de zoom, el comando no tiene efecto. Para mayor información sobre tipos de ventanas, consulte la sección .*

***Nota:** esta función no debe confundirse con la reducción de una ventana a un botón (Windows) o en el Dock (Mac OS), la cual se genera por medio de un clic en el siguiente botón:*

Windows

Mac OS

Next window

Next window (ventana) -> Resultado

Parámetro	*Tipo*		*Descripción*
ventana	*WinRef*	→	*Número de referencia de la ventana*
Resultado	*WinRef*	↩	*Número de referencia de ventana*

Descripción

El comando ***Next window*** *devuelve el número de referencia de la ventana ubicada "detrás" de la ventana que se pasa en ventana (en función del orden de las ventanas)*

Open form window

Open form window ({tabla ;} nomForm {; tipo {; posH {; posV {; *}}}}) -> Resultado

Parámetro	Tipo		Descripción
tabla	Tabla	➡	Tabla del formulario o tabla por defecto, si se omite
nomForm	Cadena, Objeto	➡	Nombre del formulario
tipo	Entero largo	➡	Tipo de la ventana
posH	Entero largo	➡	Posición horizontal de la ventana
posV	Entero largo	➡	Posición vertical de la ventana
*	Operador	➡	Conservar la posición actual y el tamaño de la ventana
Resultado	WinRef	⬅	Número de referencia de la ventana

Descripción

El comando ***Open form window*** *abre una nueva ventana utilizando las propiedades de tamaño y de redimensionamiento del formulario nomForm.*

El formulario nomForm no se muestra en la ventana. Si quiere mostrar el formulario, tiene que llamar un comando que cargue un formulario (por ejemplo ***ADD RECORD****).*

A diferencia del comando ***Open window****, ningún método está asociado a la casilla de cierre de la ventana. Al hacer clic en esta casilla se cancela y cierra la ventana, excepto si el evento de formulario On Close Box ha sido activado para el formulario. En este caso, el código asociado con este evento se ejecutará.*

Si el formulario nomForm es redimensionable, la ventana abierta tendrá una casilla de zoom como también una casilla de redimensionamiento.

Nota: *para conocer las principales propiedades de un formulario, utilice el comando* ***FORM GET PROPERTIES****.*

En el parámetro nomForm, puede pasar:

- *el nombre de un formulario (formulario proyecto o tabla) a utilizar;*
- *la ruta (en sintaxis POSIX) a un archivo .json válido que contiene una descripción del formulario a usar. Ver* ***Ruta de archivo del formulario****;*
- *un objeto que contiene una descripción del formulario a utilizar.*

El parámetro opcional tipo permite especificar un tipo de ventana. Este parámetro debe contener una de las siguientes constantes predefinidas del tema ***Abrir ventana formulario****):*

Constante	Tipo	Valor
Controller form window	Entero largo	133056
Form has full screen mode Mac	Entero largo	65536
Form has no menu bar	Entero largo	2048
Modal form dialog box	Entero largo	1
Movable form dialog box	Entero largo	5
Palette form window	Entero largo	1984
Plain form window	Entero largo	8
Pop up form window	Entero largo	32
Sheet form window	Entero largo	33
Toolbar form window	Entero largo	35

Los tipos de ventanas se detallan en la sección ***Tipos de ventanas****.*

Nota: *las constantes Form has full screen mode Mac y Form has no menu bar deben añadirse a una de las otras constantes de tipo*

De forma predeterminada, si no se pasa el parámetro tipo, se abre una ventana de tipo Plain form window.

Caja de cerrar
Las ventanas de tipo Movable form dialog box, Plain form window y Palette form window tienen una caja de cierre. No se asocia ningún método a la caja de cierre de la ventana. Al hacer clic en este caja de cierre cancela y cierra la ventana, excepto si se ha activado el evento de formulario [#cst id="845790"/] para el formulario. En este caso, se ejecutará el código asociado a este evento.

Control de tamaño
Si las propiedades de "tamaño de ventana" del formulario nomForm no están definidas como "fixed", la ventana abierta puede ser redimensionada por el usuario. Una caja de zoom puede estar disponible, dependiendo del tipo de ventana. Si la propiedad ***Ancho fijo*** *y/o* ***Altura fija*** *está marcada en las propiedades del formulario, la ventana no será redimensionable.*

Nota: *algunos atributos (caja de control de tamaño, caja cerrar ...) de la ventana creada dependen de las especificaciones de interfaz del sistema operativo para el tipo elegido. Por lo tanto, es posible obtener diferentes resultados dependiendo de la plataforma utilizada.*

El parámetro opcional posH permite definir la posición horizontal de la ventana. Puede pasar una posición definida en píxeles o una de las siguientes constantes predefinidas ubicadas en el tema ***Abrir ventana formulario****:*

Constante	Tipo	Valor
Horizontally centered	Entero largo	65536
On the left	Entero largo	131072
On the right	Entero largo	196608

El parámetro opcional posV permite definir la posición vertical de la ventana. Puede pasar una posición definida en píxeles o una de las siguientes constantes predefinidas ubicadas en el tema ***Abrir ventana formulario****:*

Constante	Tipo	Valor
At the bottom	Entero largo	393216
At the top	Entero largo	327680
Vertically centered	Entero largo	262144

Estos parámetros se expresan con relación a la esquina superior izquierda del área de contenido de la ventana de la aplicación (modo Windows MDI) o de la pantalla principal (macOS y modo Windows SDI). Tienen en cuenta la presencia de la barra de herramientas y la barra de menús.

*Si pasa el parámetro opcional *, la posición y el tamaño actual de la ventana se memorizan al cerrar. Cuando la ventana se abre nuevamente, se respetan su posición y tamaño anterior. En este caso, los parámetros posV y PosH sólo se utilizan cuando se abre la ventana por primera vez.*

***Nota:** para reabrir una ventana con sus coordenadas por defecto cuando el parámetro vPos y hPos se pasa, mantenga presionada la tecla **Mayús** mientras se abre la ventana.*

Ejemplo 1

La siguiente instrucción abre una ventana estándar o una casilla cerrar y se ajusta automáticamente al tamaño del formulario de "Entrada". Como el formulario ha sido definido como redimensionables, la ventana tiene una casilla de redimensionamiento y una casilla de zoom:

```
$winRef :=Open form window([Table1];"Enter")
```

Ejemplo 2

La siguiente instrucción abre una paleta flotante en la parte superior izquierda de la pantalla basada en un formulario de proyecto llamado "Herramientas". Esta paleta utiliza la última posición en cada nueva apertura:

```
$winRef :=Open form window("Herramientas";Palette form window;On the left;At the top;*)
```

Ejemplo 3

Este código se debe llamar en macOS mientras se muestra una ventana documento, por ejemplo desde un botón de formulario, para mostrar una ventana hoja:

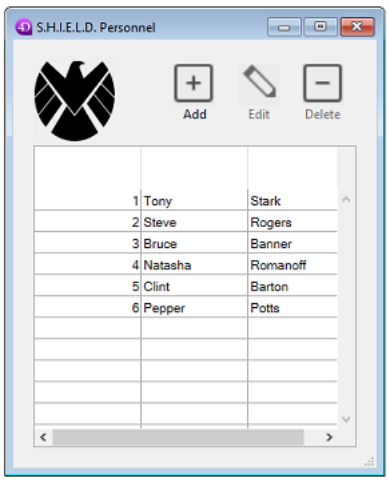
```
$myWin:=Open form window("sheet form";Sheet form window)
 // Por el momento, la ventana se crea pero permanece oculta
DIALOG([aTable];"dialForm")
 //El evento On Load se genera luego se muestra la ventana hoja; "Cae"
 //desde la parte inferior de la barra de título
```

Ejemplo 4

El siguiente ejemplo usa la ruta a un formulario .json para mostrar los registros en una lista de empleados:

```
Open form window("/RESOURCES/OutputPersonnel.json";Plain form window)
ALL RECORDS([Personnel])
DIALOG("/RESOURCES/OutputPersonnel.json";*)
```

que devuelve:



Open window

Open window (izquierda ; superior ; derecha ; inferior {; tipo {; titulo {; casillaCerrar}}}) -> Resultado

Parámetro	Tipo		Descripción
izquierda	Entero largo	➡	Coordenada global izquierda del interior de la ventana
superior	Entero largo	➡	Coordenada global superior del interior de la ventana
derecha	Entero largo	➡	Coordenada global derecha del interior de la ventana
inferior	Entero largo	➡	Coordenada global inferior del interior de la ventana
tipo	Entero largo	➡	Tipo de la ventana
titulo	Cadena	➡	Título de la ventana
casillaCerrar	Cadena	➡	Método a llamar en caso de doble clic del menú Control o de clic en la casilla Cerrar
Resultado	WinRef	⬅	Número de referencia de la ventana

Descripción

Open window *abre una nueva ventana cuyas dimensiones son definidas por los cuatro primeros parámetros:*

- *izquierda es la distancia en píxeles desde el lado izquierdo de la ventana de la aplicación y el costado interior izquierdo de la ventana.*
- *superior es la distancia en píxeles entre la altura de la ventana de la aplicación y el borde superior del interior de la ventana.*
- *derecha es la distancia en píxeles entre el lado izquierdo de la ventana de la aplicación y el costado interior derecho de la ventana.*
- *inferior es la distancia en píxeles desde la parte superior de la ventana de la aplicación y borde interior inferior de la ventana.*

Nota de compatibilidad*:* ***Open window*** *integra diferentes opciones que han evolucionado a través de las versiones y ahora sólo se conserva por razones de compatibilidad. Cuando se escribe nuevo código para la gestión de ventanas, por lo general es más práctico utilizar el comando* ***Open form window****, que se adapta mejor a las interfaces actuales.*

Si pasa -1 en derecha e inferior, le indica a 4D que redimensione automáticamente la ventana si se cumplen las siguientes condiciones:

- *Usted ha diseñado un formulario y definido sus opciones de redimensionamiento en la ventana de propiedades del formulario en el entorno Diseño*
- *Antes de llamar* ***Open window****, usted seleccionó el formulario utilizando el comando* ***FORM SET INPUT****, al cual pasó el parámetro opcional *.*

Importante: *este dimensionamiento automático de la ventana ocurrirá únicamente si realiza una llamada previa a* ***FORM SET INPUT*** *para el formulario a mostrar en la ventana y si le pasa el parámetro opcional * a* ***FORM SET INPUT****.*

- *El parámetro tipo es opcional y define el tipo de ventana que quiere mostrar, y corresponde a las diferentes ventanas presentadas en la sección . Si el tipo pasado es negativo, la ventana será flotante. Si el tipo no se especifica, el tipo 1 se utiliza por defecto.*
- *El parámetro titulo indica el título opcional de la ventana*

Si pasa una cadena vacía ("") en titulo, le indica a 4D que utilice los valores de introducidos en el área Nombre de la ventana de la ventana de Propiedades del formulario en el entorno Diseño para el título del formulario a mostrar en la ventana.

Importante: *el título por defecto del formulario se aplicará a la ventana únicamente se llama previamente al comando* ***FORM SET INPUT*** *para el formulario a mostrar y le pasa el parámetro opcional * a* ***FORM SET INPUT****.*

- *El parámetro casillaCerrar es opcional y designa el método para cerrar la ventana. Si se especifica este parámetro, la casilla del menú Control (Windows) o la casilla Cerrar (Macintosh) se añade a la ventana. Cuando el usuario hace doble clic en la casilla de menú Control (Windows) o clic en la Casilla cerrar (Macintosh), se llama al método pasado en casillaCerrar.*

Nota: *también puede administrar el cierre de la ventana desde el método del formulario mostrado en la ventana cuando ocurre un evento On Close Box. Para mayor información, consulte el comando* ***Evento formulario*** *.*

Si se abre más de una ventana para un proceso, la última ventana abierta es la ventana activa (del primer plano) para ese proceso. Sólo puede modificarse la información dentro de la ventana activa. Todas las demás ventanas pueden ser visualizadas. Cuando el usuario digita, la ventana activa siempre pasará al primer plano, si aún no está ahí.

Los formularios se muestran al interior de una ventana abierta. El texto pasado al comando ***MESSAGE*** *también aparece en la ventana.*

Open window *devuelve una referencia de ventana de tipo WinRef, utilizable por los comandos de gestión de ventanas (ver la sección "****WinRef****").*

Ejemplo 1

El siguiente método de proyecto abre una ventana centrada en la ventana en la ventana principal (Windows) o en la pantalla principal (Macintosh). Note que puede aceptar dos, tres, o cuatro parámetros:

```
` Método de proyecto OPEN CENTERED WINDOW
` $1 – Ancho de la ventana
` $2 – Alto de la ventana
` $3 – Tipo de la ventana (opcional)
` $4 – Título de la ventana (opcional)
```

```
$SW:=Screen width\2
$SH:=(Screen height\2)
$WW:=$1\2
$WH:=$2\2
Case of
   :(Count parameters=2)
      Open window($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH)
   :(Count parameters=3)
      Open window($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3)
   :(Count parameters=4)
      Open window($SW-$WW;$SH-$WH;$SW+$WW;$SH+$WH;$3;$4)
End case
```

Una vez escrito el método de proyecto, puede utilizarlo de esta forma:

```
OPEN CENTERED WINDOW(400;250;Movable dialog box;"Actualizar archivos")
DIALOG([Tabla de herramientas];"UPDATE OPTIONS")
CLOSE WINDOW
If(OK=1)
   ` ...
End if
```

Ejemplo 2

El siguiente ejemplo abre una ventana flotante que tiene un casilla de menú Control (Windows) o una casilla de cerrar (Macintosh). La ventana se abre en la esquina superior derecha de la ventana de la aplicación.

```
$myWindow:=Open window(Screen width-149;33;Screen width-4;178;-Palette window;"";"CloseColorPalette")
DIALOG([Dialogs];"Color Palette")
```

*El método **CloseColorPalette** llama al comando **CANCEL**:*

```
CANCEL
```

Ejemplo 3

El siguiente ejemplo abre una ventana cuyo tamaño y título provienen de las propiedades del formulario mostrado en la ventana:

```
ORM SET INPUT([Customers];"Add Records";*)
$myWindow:=Open window(10;80;-1;-1;Plain window;"")
Repeat
   ADD RECORD([Customers])
Until(OK=0)
```

***Recuerde**: para que la función **Open window** utilice automáticamente las propiedades del formulario, debe llamar a **FORM SET INPUT** con el parámetro opcional * y las propiedades del formulario deben haber sido definidas en función de esta utilización en el entorno Diseño.*

Ejemplo 4

Este ejemplo ilustra el mecanismo de "retraso" de mostrar ventanas bajo Mac OS X:

```
$miVentana:=Open window(10;10;400;400;Sheet window)
   `Por el momento, se crea la ventana pero permanece oculta
DIALOG([Tabla];"formDial")
   `El evento On Load se genera luego se muestra la ventana; "desciende" de debajo de la barra de título
```

REDRAW WINDOW

REDRAW WINDOW {(ventana)}

Parámetro	*Tipo*		*Descripción*
ventana	*WinRef*	⇒	*Número de referencia de la ventana o Ventana del primer plano del proceso actual, si se omite*

Descripción

*El comando **REDRAW WINDOW** provoca una actualización del contenido de la ventana cuyo número de referencia se pasa en ventana.*

*Si omite el parámetro ventana, **REDRAW WINDOW** aplica a la ventana del primer plano del proceso actual.*

***Nota:** 4D administra automáticamente las actualizaciones gráficas de las ventanas cada vez que usted mueve, redimensiona o pasa al primer plano una ventana, así como también cuando usted cambia el formulario y/o los valores mostrados en la ventana. Este comando se utiliza con muy poca frecuencia.*

RESIZE FORM WINDOW

RESIZE FORM WINDOW (ancho ; alto)

Parámetro	Tipo		Descripción
ancho	Entero largo	➡	Píxeles a añadir o eliminar del ancho actual de la ventana formulario
alto	Entero largo	➡	Píxeles a añadir o eliminar del largo actual de la ventana formulario

Descripción

*El comando **RESIZE FORM WINDOW** permite modificar el tamaño de la ventana del formulario actual.*

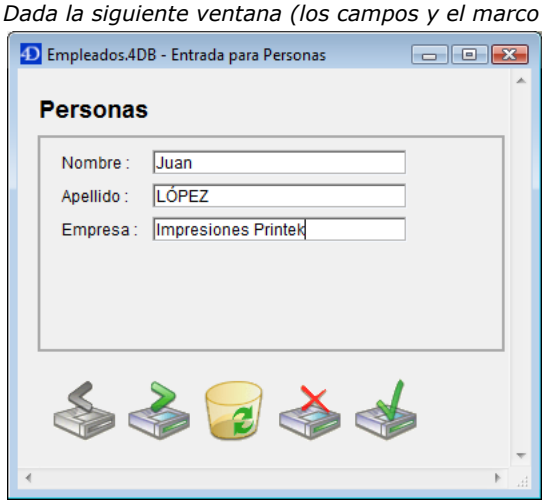
Pase el número de píxeles que quiere añadir al tamaño de la ventana actual en los parámetros ancho y alto. Pase 0 en el parámetro que no quiera modificar. Para reducir el tamaño, pase un valor negativo en los parámetros ancho y alto.

Este comando produce exactamente el mismo resultado que un redimensionamiento manual utilizando la casilla de redimensionamiento (si el tipo de ventana lo permite). Por consiguiente, el comando tiene en cuenta las propiedades de redimensionamiento de los objetos y las limitaciones de tamaño definidas en las propiedades del formulario. Si, por ejemplo, el comando redimensiona la ventana a un tamaño superior al máximo del formulario, el comando no tiene efecto.

*Por favor tenga en cuenta que este comportamiento es diferente del comportamiento del comando **SET WINDOW RECT**, el cual no tiene en cuenta las propiedades del formulario ni su contenido cuando redimensiona la ventana. Igualmente, note que este comando no necesariamente modifica el tamaño del formulario. Para modificar el tamaño de un formulario por programación, por favor consulte la descripción del comando **FORM SET SIZE**.*

Ejemplo

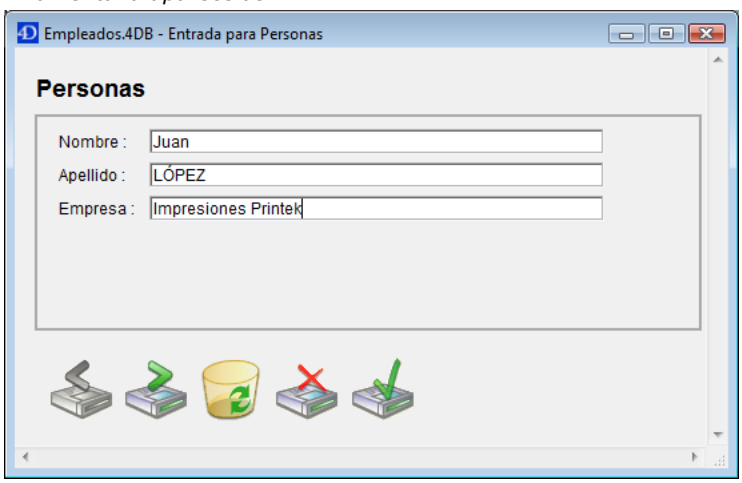
Dada la siguiente ventana (los campos y el marco tienen la propiedad de dimensionamiento horizontal "Agrandar"):



Después de la ejecución de esta línea:

```
RESIZE FORM WINDOW(25;0)
```

... la ventana aparece así:



SET WINDOW RECT

SET WINDOW RECT (izquierda ; superior ; derecha ; inferior {; ventana}{; *})

Parámetro	Tipo		Descripción
izquierda	Entero largo	➡	Coordenada global izquierda del interior de la ventana
superior	Entero largo	➡	Coordenada global superior del interior de la ventana
derecha	Entero largo	➡	Coordenada global derecha del interior de la ventana
inferior	Entero largo	➡	Coordenada global inferior del interior de la ventana
ventana	WinRef	➡	Número de referencia de la ventana o ventana del primer plano del proceso si se omite este parámetro
*	Operador	➡	Si se omite (por defecto) = cambiar ventana al fondo Si se pasa = no cambiar el nivel de la ventana

Descripción

*El comando **SET WINDOW RECT** cambia las coordenadas globales de la ventana cuyo número de referencia se pasa en ventana. Si la ventana no existe, el comando no hace nada.*

*Si omite el parámetro ventana, **SET WINDOW RECT** se aplica a la ventana del primer plano del proceso actual.*

Este comando puede redimensionar y mover la ventana, dependiendo de las nuevas coordenadas que se pasen.

Las coordenadas deben ser expresadas con relación a la esquina superior izquierda del área de contenido de la ventana de la aplicación (modo Windows MDI) o de la pantalla principal (mac OS y modo Windows MDI). Las coordenadas indican el rectángulo correspondiente al área de contenido de la ventana (excluyendo las barras de títulos y los bordes).

***Advertencia:** utilice este comando con precaución, podría mover una ventana más allá de los límites de la ventana principal (en Windows) o de la pantalla (en Macintosh). Para evitar esto, utilice comandos como **Screen width** y **Screen height** para verificar las nuevas coordenadas de la ventana.*

*Por defecto, la ejecución de este comando pasa al primer plano la ventana designada por el parámetro ventana (si se utiliza este parámetro). Puede desactivar este funcionamiento pasando * como último parámetro. En este caso, el comando no modifica el nivel original de la ventana (coordenada "z").*

*Este comando no afecta los objetos de formulario. Si la ventana contiene un formulario, los objetos del formulario no se mueven o redimensionan por el comando (sin importar sus propiedades). Sólo se modifica la ventana. Para modificar una ventana de formulario teniendo en cuenta sus propiedades de redimensionamiento y los objetos que contiene, debe utilizar el comando **RESIZE FORM WINDOW**.*

Ejemplo 1

*Ver el ejemplo del comando **WINDOW LIST**.*

Ejemplo 2

Dada la siguiente ventana:

Después de la ejecución de la siguiente línea:

```
SET WINDOW RECT(100;100;300;300)
```

La ventana aparece de esta forma:


Empleados.4D...
Personas
Nombre
Juan
Apellido
LÓPEZ
Empresa
Impresiones F

SET WINDOW TITLE

SET WINDOW TITLE (titulo {; ventana})

Parámetro	Tipo		Descripción
titulo	Cadena	➡	Título de la ventana
ventana	WinRef	➡	Número de referencia de ventana o Ventana del primer plano del proceso actual si se omite el parámetro

Descripción

*El comando **SET WINDOW TITLE** cambia el título de la ventana cuyo número de referencia se pasa en ventana para el texto pasado en titulo (longitud máxima 80 caracteres).*

*Si la ventana no existe, **SET WINDOW TITLE** no hace nada.*

*Si omite el parámetro ventana, **SET WINDOW TITLE** reemplaza el título de la ventana del primer plano del proceso actual.*

***Nota:** en el entorno Diseño, 4D define automáticamente los títulos de las ventanas, por ejemplo, "Entrada para tabla 1" cuando realiza una entrada de datos. Si cambia un título de ventana, es probable que 4D lo sobrescriba. Por otra parte, en el entorno Aplicación, 4D no cambia los títulos de las ventanas.*

Ejemplo

Mientras efectúa una entrada de datos en un formulario, usted hace clic en un botón que ejecuta una operación larga (por ejemplo, una modificación por programación de los registros relacionados mostrados en un subformulario). Puede mantenerse informado sobre el progreso de la operación utilizando el título de la ventana actual:

```
` Método de objeto del botón bAnalisis
Case of
   :(Form event=On Clicked)
   ` Guardar el título actual de la ventana en una variable
      $vsTituloActual:=Get window title
   ` Iniciar la operación larga
      FIRST RECORD([Lineas factura])
      For($vlRegistro;1;Records in selection([Lineas factura]))
         HACER ALGO
   ` Mostrar el progreso
         SET WINDOW TITLE("Procesando la línea #"+String($vlRegistro))
      End for
   ` Restaurar el título original de la ventana
      SET WINDOW TITLE($vsTituloActual)
End case
```

SHOW TOOL BAR

SHOW TOOL BAR
Este comando no requiere parámetros

Descripción

*El comando **SHOW TOOL BAR** maneja la visualización de las barras de herramientas personalizadas creadas por el comando **Open form window** para el proceso actual.*
*Si una ventana barra de herramientas ha sido creada por el comando **Open form window** con la opción Toolbar form window, el comando hace esta ventana visible. Si la ventana barra de herramientas ya era visible o si ninguna ventana de este tipo ha sido creada, el comando no hace nada.*

Ejemplo

*Consulte el ejemplo del comando **HIDE TOOL BAR**.*

SHOW WINDOW

SHOW WINDOW {(ventana)}

Parámetro	Tipo		Descripción
ventana	WinRef	➡	Número de referencia de la ventana o Ventana del primer plano del proceso actual, si se omite

Descripción

*El comando **SHOW WINDOW** permite mostrar la ventana cuyo número se pasó en ventana. Si se omite este parámetro, se mostrará la ventana del primer plano del proceso actual.*

*Para utilizar el comando **SHOW WINDOW**, la ventana debe haberse ocultado utilizando el comando **HIDE WINDOW**. Si la ventana ya es visible, el comando no hace nada.*

Ejemplo

*Consulte el ejemplo del comando **HIDE WINDOW**.*

Tool bar height

Tool bar height -> Resultado

Parámetro	*Tipo*		*Descripción*
Resultado	*Entero largo*	➲	*Altura (expresada en píxeles) de la barra de herramientas o 0 si la barra está oculta*

Descripción

*El comando **Tool bar height** devuelve la altura de la barra de herramientas visible actualmente, expresada en píxeles. Dependiendo del contexto, puede ser la barra de herramientas de modo Diseño 4D, o una barra de herramientas personalizada creada con **Open form window** (la barra de herramientas de modo Diseño se oculta automáticamente cuando se muestra una barra de herramientas personalizada).*

Si no se muestra ninguna barra de herramientas, el comando devuelve 0.

Window kind

Window kind {(ventana)} -> Resultado

Parámetro	Tipo		Descripción
ventana	WinRef	→	Número de referencia de la ventana o Ventana del primer plano del proceso actual, si se omite
Resultado	Entero largo	←	Tipo de ventana

Descripción

*El comando **Window kind** devuelve el tipo de ventana 4D cuyo número de referencia se pasa en ventana. Si la ventana no existe, **Window kind** devuelve 0 (cero).*

*De lo contrario, **Window kind** devuelve una de las siguientes constantes predefinidas (tema **Ventana**):*

Constante	Tipo	Valor
External window	Entero largo	5
Floating window	Entero largo	14
Modal dialog	Entero largo	9
Regular window	Entero largo	8

*Si omite el parámetro ventana, **Window kind** devuelve el tipo de la ventana del primer plano del proceso actual.*

Ejemplo

*Ver el ejemplo del comando **WINDOW LIST**.*

WINDOW LIST

WINDOW LIST (ventanas {; *})

Parámetro	Tipo		Descripción
ventanas	Array	⇐	Array de los números de referencia de las ventanas
*	Operador	⇒	Si se especifica, tiene en cuenta las ventanas flotantes Si se omite, ignora las ventanas flotantes

Descripción

*El comando **WINDOW LIST** llena el array ventanas con los números de referencia de las ventanas abiertas actualmente en todos los procesos (procesos kernel o usuario). Sólo las ventanas "visibles" (ventanas no ocultas) se devuelven.*

*Si no pasa el parámetro opcional *, se ignoran las ventanas flotantes.*

Ejemplo

El siguiente método de proyecto coloca en mosaico todas las ventanas abiertas actualmente, excepto las ventanas flotantes y las cajas de diálogo:

```
` Método de proyecto TILE WINDOWS

WINDOW LIST($alWnd)
$vlLeft:=10
$vlTop:=80 ` Dejar espacio para la barra de herramientas
For($vlWnd;1;Size of array($alWnd))
   If(Window kind($alWnd{$vlWnd})#Modal dialog)
      GET WINDOW RECT($vlWL;$vlWT;$vlWR;$vlWB;$alWnd{$vlWnd})
      $vlWR:=$vlLeft+($vlWR-$vlWL)
      $vlWB:=$vlTop+($vlWB-$vlWT)
      $vlWL:=$vlLeft
      $vlWT:=$vlTop
      SET WINDOW RECT($vlWL;$vlWT;$vlWR;$vlWB;$alWnd{$vlWnd})
      $vlLeft:=$vlLeft+10
      $vlTop:=$vlTop+25
   End if
End for
```

***Nota:** este método puede mejorarse añadiendo pruebas del tamaño de la ventana principal (en Windows) o del tamaño y ubicación de las pantallas (en Macintosh).*

Window process

Window process {(ventana)} -> Resultado

Parámetro	*Tipo*		*Descripción*
ventana	*WinRef*	→	*Número de referencia de la ventana*
Resultado	*Entero largo*	↩	*Número de referencia del proceso*

Descripción

El comando ***Window process*** *devuelve el número de proceso que ejecuta la ventana cuyo número de referencia se pasa en ventana. Si la ventana no existe, el comando devuelve 0 (cero).*

Si omite el parámetro ventana, ***Window process*** *devuelve el número del proceso de la ventana del primer plano del proceso actual.*

_o_Open external window

_o_Open external window (izquierda ; superior ; derecha ; inferior ; tipo ; titulo ; areaPlugin) -> Resultado

Parámetro	Tipo		Descripción
izquierda	Entero largo	➡	Coordenada izquierda global de la ventana
superior	Entero largo	➡	Coordenada superior global de la ventana
derecha	Entero largo	➡	Coordenada derecha global de la ventana
inferior	Entero largo	➡	Coordenada inferior global de la ventana
tipo	Entero largo	➡	Tipo de ventana
titulo	Cadena	➡	Título de la ventana
areaPlugin	Cadena	➡	Comando de área externa
Resultado	WinRef	↩	Número de referencia de la ventana y del área externa

Descripción

*El comando **_o_Open external window** se declara obsoleto en 4D a partir de la versión 16 y solo se mantiene por razones de compatibilidad. Note que no es compatible con versiones 64 bits de 4D.*

Tipos de ventanas (Compatibilidad)

Nota de compatibilidad

Algunos tipos de ventana descritos en esta sección están relacionados con las versiones anteriores de 4D y del sistema operativo. Este tema, así como también el comando ***Open window*** *solo se mantienen ahora por razones de compatibilidad. Cuando escriba un nuevo código para administrar ventanas, recomendamos encarecidamente el uso del comando* ***Open form window****, que se adapta mejor a las interfaces actuales.*

Presentation

*Puede utilizar una de las siguientes constantes predefinidas (tema "****Crear ventana****") para especificar el tipo de ventana a abrir con* ***Open window****:*

Constante	Tipo	Valor	Comentario
Plain no zoom box window	*Entero largo*	*0*	
Modal dialog box	*Entero largo*	*1*	
Plain dialog box	*Entero largo*	*2*	*Puede utilizarse en ventana flotante*
Alternate dialog box	*Entero largo*	*3*	*Puede utilizarse en ventana flotante*
Plain fixed size window	*Entero largo*	*4*	
Movable dialog box	*Entero largo*	*5*	*Puede utilizarse en ventana flotante*
Plain window	*Entero largo*	*8*	
Round corner window	*Entero largo*	*16*	
Pop up window	*Entero largo*	*32*	
Sheet window	*Entero largo*	*33*	
Resizable sheet window	*Entero largo*	*34*	
Palette window	*Entero largo*	*1984*	*Puede utilizarse en ventana flotante*

Ventanas flotantes

Si pasa una de estas constantes a ***Open window****, abre una ventana estándar. Para abrir una ventana flotante, pase un tipo de ventana negativo a* ***Open window****.*

La principal característica de las ventanas flotantes es que permanezcan en primer plano, incluso si el usuario hace clic en otra ventana del proceso. Las ventanas flotantes se utilizan generalmente para mostrar información permanente o barras de herramientas.

Ventanas modales

Una ventana modal coloca al usuario en un estado (o "modo") donde sólo puede actuar dentro de esta ventana. Mientras se muestre la ventana modal, los comandos de menú y las otras ventanas de la aplicación son inaccesibles. Para cerrar una ventana modal, el usuario debe validarla, cancelarla, o elegir una de las opciones que ofrece. Las cajas de diálogo de alerta son ejemplos típicos de ventanas modales.

En 4D, las ventanas de tipo 1 y 5 son ventanas modales.

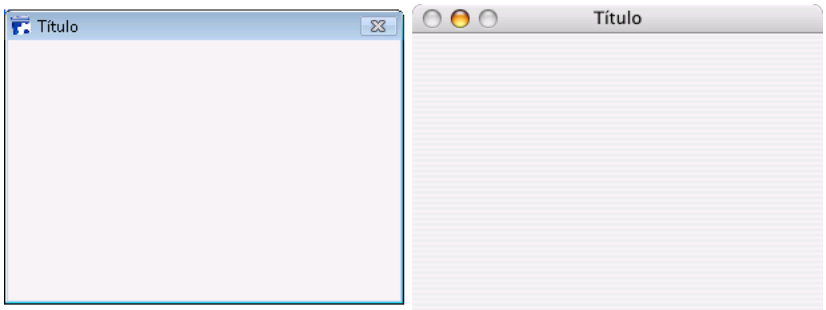
Nota: *una ventana modal siempre permanece en el primer plano. Por consiguiente, cuando una ventana modal llama a una ventana no modal, esta última ventana se muestra al fondo, incluso si se llamó después de la ventana modal. Por lo tanto evite este tipo de operación.*

Por el contrario, cuando una ventana modal llama a otra ventana modal, la última ventana se mostrará en el primer plano.

Descripción

La siguiente muestra cada tipo de ventana, en Windows (izquierda) y en Macintosh (derecha).

Plain fixed size window (4)

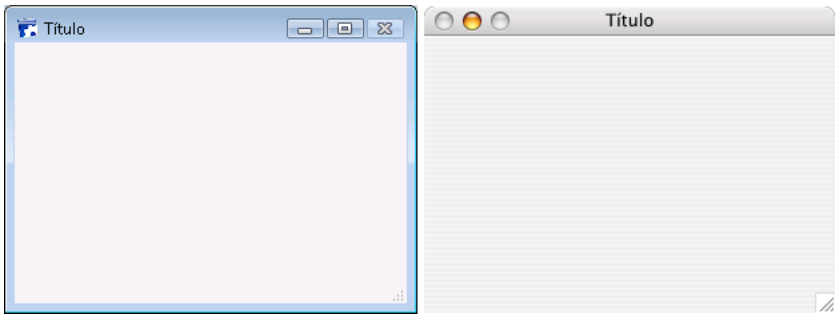


- *Puede tener un título: sí*
- *Puede tener una casilla Cerrar o un equivalente: sí*
- *Puede redimensionarse: no en Macintosh*
- *Puede minimizarse/maximizarse o hacerle zoom: no*
- *Adaptada a las barras de desplazamiento: sí y no*
- *Uso: entrada de datos con* ***ADD RECORD*** *o equivalente*

Modal dialog box (1)

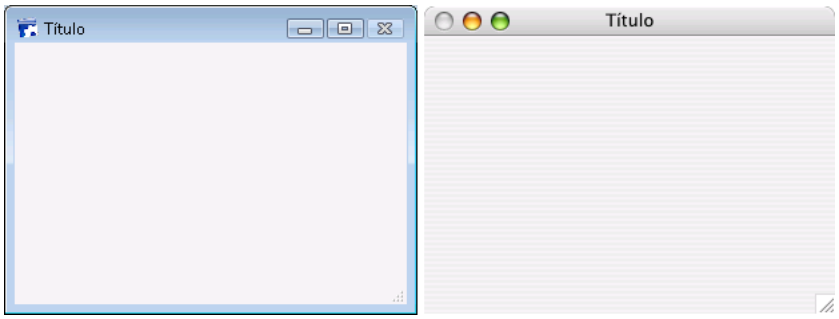
- *Puede tener un título: no*
- *Puede tener una casilla Cerrar o un equivalente: no*
- *Puede redimensionarse: no*
- *Puede minimizarse/maximizarse o hacerle zoom: no*
- *Adaptada a las barras de desplazamiento: no*
- *Uso: **DIALOG**, **ADD RECORD** o equivalente*
- *Las ventanas de este tipo son modales*

Plain no zoom box window (0)



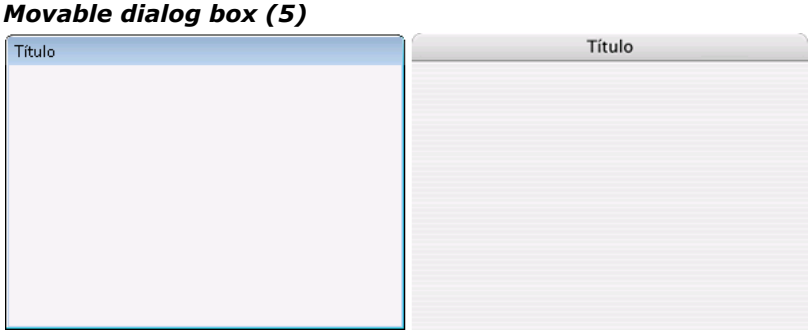
- *Puede tener un título: sí*
- *Puede tener una casilla Cerrar o un equivalente: sí*
- *Puede redimensionarse: sí*
- *Puede minimizarse/maximizarse o hacerle zoom: no en Macintosh*
- *Adaptada a las barras de desplazamiento: sí*
- *Uso: entrada de datos con barras de desplazamiento, **DISPLAY SELECTION**, **MODIFY SELECTION**, etc.*

Plain window (8)



- *Puede tener un título: sí*
- *Puede tener una casilla Cerrar o un equivalente: sí*
- *Puede redimensionarse: sí*
- *Puede minimizarse/maximizarse o hacerle zoom: sí*
- *Adaptada a las barras de desplazamiento: sí*
- *Uso: entrada de datos con barras de desplazamiento, **DISPLAY SELECTION**, **MODIFY SELECTION**, etc.*

Movable dialog box (5)



- *Puede tener un título: sí*
- *Puede tener una casilla Cerrar o un equivalente: no*
- *Puede redimensionarse: no*
- *Puede minimizarse/maximizarse o hacerle zoom: no*
- *Adaptada a las barras de desplazamiento: no*

- *Uso: **DIALOG, ADD RECORD**(...;...;*) o equivalente*
- *Las ventanas de este tipo son modales, pero pueden moverse para utilizarse como ventanas flotantes*

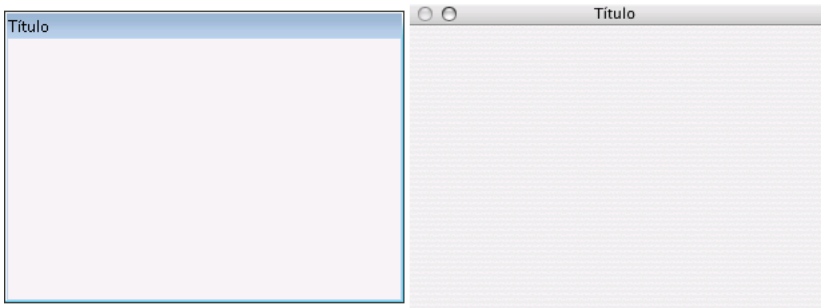
Alternate dialog box (3)

- *Puede tener un título: no*
- *Puede tener una casilla Cerrar o un equivalente: no*
- *Puede ser redimensionada: no*
- *Puede ser minimizada/maximizada o hacerle zoom: no*
- *Adaptada a las barras de desplazamiento: no*
- *Uso: **DIALOG**, **ADD RECORD**(...;...;*) o equivalente*
- *Las ventanas de este tipo son modales, a menos que se utilicen como ventanas flotantes*

Plain dialog box (2)

- *Puede tener un título: no*
- *Puede tener una casilla Cerrar o un equivalente: no*
- *Puede redimensionarse: no*
- *Puede minimizarse/maximizarse o hacerle zoom: no*
- *Adaptada a las barras de desplazamiento: no*
- *Uso: **DIALOG**, **ADD RECORD**(...;...;*) o equivalente, bajo Mac OS (no estándar bajo Windows)*
- *Las ventanas de este tipo son modales, a menos que se utilicen como ventanas flotantes*

Ventana Paleta (1984)



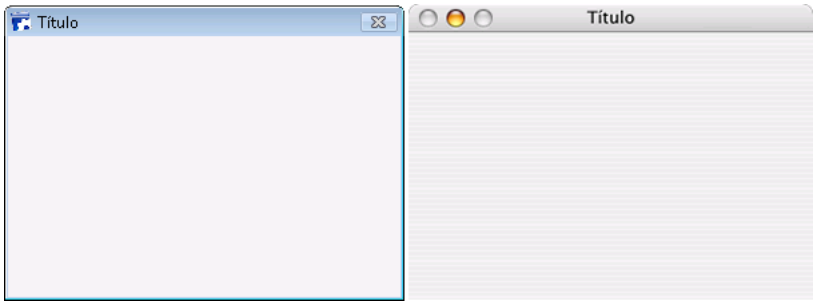
Este tipo de ventana le permite generar ventanas flotantes que se pueden definir como redimensionables o no. Sólo se admiten las siguientes opciones:

Opción	***Valor a pasar bajo Windows***	***Valor a pasar bajo macOS***
No redimensionable	*-(Palette window+2)*	*-Palette window*
Redimensionable	*-(Palette window+6)*	*-(Palette window+6)*

- *Puede tener un título: sí, si se pasa*
- *Puede redimensionarse: sí, si se pasa el valor apropiado*
- *Uso: ventanas flotantes con **DIALOG** o **DISPLAY SELECTION** (sin entrada de datos).*

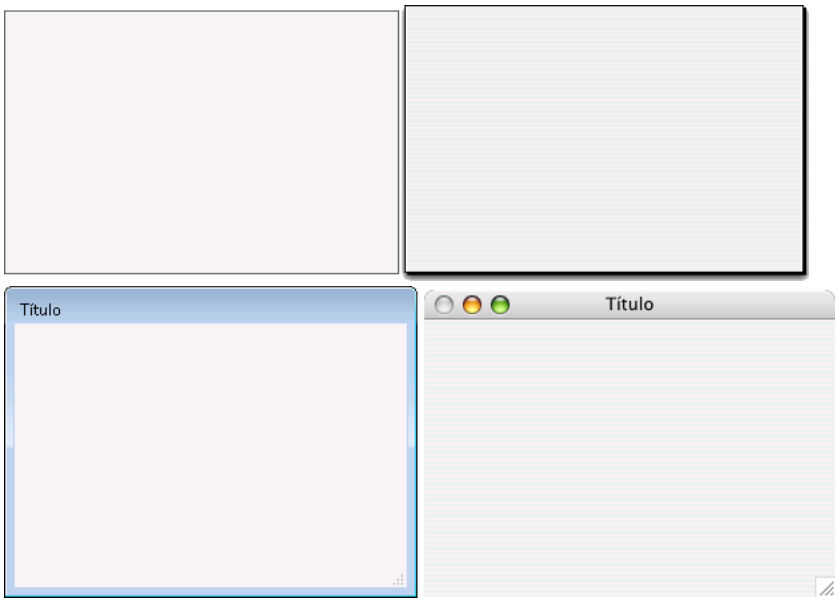
***Nota:** con este tipo de ventana, el conjunto de valores (opción + constante) siempre se debe pasar como un valor negativo. Asegúrese de pasar, por ejemplo, -(Palette window+6) y no (-Palette window+6).*

Round corner window (16)



- *Puede tener un título: sí*
- *Puede tener una casilla Cerrar o un equivalente: sí*
- *Puede redimensionarse: no en Macintosh*
- *Puede minimizarse/maximizarse o hacerle zoom: No*
- *Adaptada a las barras de desplazamiento: No en Macintosh*
- *Uso: raro (obsoleto)*

Sheet window (33) and Resizable sheet window (34)



Las ventanas hoja (sheet windows) son específicas para Mac OS X. Estas ventanas "descienden" de la barra de título de la ventana principal utilizando una animación y se muestran sobre la ventana principal. Se centran automáticamente en la ventana principal. Sus propiedades son idénticas a las de las cajas de diálogo modales. Por lo general se utilizan para realizar una acción directamente relacionada con la que se lleva a cabo en la ventana principal.

- *Puede crear una ventana hoja únicamente bajo Mac OS X si la última ventana abierta es visible y de tipo documento (form).*
- *El comando abre una ventana de tipo 1 (diálogo modal) en lugar de una de tipo 33 o de tipo 8 (ventana estándar) en lugar de una de tipo 34:*
  - si la última ventana abierta no es visible o no es de tipo documento,
  - bajo Windows.
- *Como una ventana hoja debe dibujarse sobre un formulario, su visualización se rechaza en el evento On load del primer formulario cargado en la ventana (ver el ejemplo 4 del comando **Open window**).*
- *Uso: **DIALOG**, **ADD RECORD**(...;...*) o equivalente, bajo Mac OS (no estándar bajo Windows).*

Pop up window (32)

Este tipo de ventana tiene las características esenciales del tipo Diálogo simple (2) y dispone de propiedades avanzadas específicas:

- *La ventana se cierra automáticamente y el evento "cancelar" se pasa a la ventana cuando:*
  - ocurre un clic fuera de la ventana;
  - la ventana de fondo o la ventana MDI (Multiple Document Interface) se desplaza;
  *- el usuario hace clic en la tecla **Esc**.*
- *Esta ventana se muestra delante de su ventana "padre" (no debe utilizarse como ventana principal del proceso). La ventana de fondo no está desactivada. Sin embargo, no recibe más eventos.*
- *No es posible redimensionar o desplazar la ventana utilizando el ratón; sin embargo, al realizar estas acciones por programación, se optimiza el redimensionamiento de los elementos del fondo.*

- *Uso: este tipo de ventana se adapta particularmente al manejo de los menús pop-up asociados a los botones 3D de tipo "bevel" o "barras de herramientas".*
- *Limitaciones:*
  - *No es posible mostrar objetos de menú pop-up dentro de este tipo de ventana.*
  - *A partir de la versión 13 de 4D, este tipo de ventana no permite mostrar mensajes de ayuda en Mac OS.*

Texture appearance (2048)

Bajo Mac OS, es posible aplicar el aspecto metálico a las ventanas. Este tipo de apariencia se encuentra con frecuencia en la interfaz Macintosh. Bajo Windows, esta propiedad no tiene efecto.

Para aplicar la apariencia metálica a una ventana creada por el comando ***Open window****, puede añadir la constante Texture appearance al tipo de ventana definido en el parámetro tipo. Por ejemplo:*

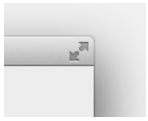
```
$ven:=Open window(10;80;-1;-1;Plain window+Texture appearence;"")
```

Esta apariencia está asociada a los siguientes tipos de ventanas:

Ventana estándar
Ventana estándar sin zoom
Ventana estándar de tamaño fijo
Diálogo modal desplazable
Ventana de esquinas redondeadas

Con modo pantalla completa Mac (65536)

La opción de "pantalla completa" está disponible a partir de 4D v14 en OS X para las ventanas de tipo de documento. Cuando se utiliza esta opción, el botón "Pantalla completa" se muestra en la esquina superior derecha de la ventana:



Cuando el usuario hace clic en este icono, la ventana pasa a pantalla completa y 4D oculta automáticamente la barra de herramientas principal.
Para utilizar esta opción, añada la constante Has full screen mode Mac al parámetro tipo para los comandos ***Open window****,* ***Open form window*** *y* ***_o_Open external window****. Por ejemplo, este código crea una ventana formulario con botón pantalla completa bajo OS X:*

```
$win:=Open form window([Interface];"User_Choice";Plain form window+Form has full screen mode Mac)
DIALOG([Interface];"User_Choice")
```

Nota: *bajo Windows, esta opción no tiene efecto.*

XML

Presentación de los comandos XML utilitarios
XML DECODE
XML GET ERROR
XML GET OPTIONS
XML SET OPTIONS
_o_XSLT APPLY TRANSFORMATION
_o_XSLT GET ERROR
_o_XSLT SET PARAMETER

Presentación de los comandos XML utilitarios

Este tema agrupa los comandos XML genéricos "utilitarios" de 4D. Estos son comandos de gestión de errores y de opciones como también comandos especializados en XSL.

Para información general sobre XML (presentación, glosario) como también las diferencias entre los modos DOM y SAX, consulte la sección ***Presentación de los comandos XML DOM****.*

¿Qué es SVG?

SVG (Scalable Vector Graphics) es un formato de archivo utilizado para describir en XML vectores gráficos (extensión .svg). El uso más común de SVG es la publicación de datos estadísticos o cartográficos.
Estos archivos pueden visualizarse en navegadores Web, bien sea nativamente, vía los plug-ins. 4D v11 incluye un motor de renderización SVG que le permite visualizar los archivos SVG en los campos o las variables imagen. El comando ***SVG EXPORT TO PICTURE*** *permite generar una imagen en 4D a partir de una descripción SVG. Igualmente note que el comando* ***GRAPH*** *puede utilizarse para aprovechar el motor SVG integrado de 4D.*
Para mayor información sobre este formato, consulte la siguiente dirección: http://www.w3.org/Graphics/SVG/.

Sobre los comandos XSLT

A partir de 4D v14 R4, los comandos de transformación XSL se declaran obsoletos y se les coloca un prefijo en consecuencia:

Nombre anterior	**4D v14 R4 y superiores**
XSLT APPLY TRANSFORMATION	**_o_XSLT APPLY TRANSFORMATION**
XSLT GET ERROR	**_o_XSLT GET ERROR**
XSLT SET PARAMETER	**_o_XSLT SET PARAMETER**

Por compatibilidad, las transformaciones XSL siguen siendo soportadas por 4D, pero su uso no se recomienda. En futuras versiones de 4D, ya no será posible utilizar la tecnología XSLT.

Nota para 4D Server 64-bit OS X: *XSLT no está incluido en la versión 4D Server 64 bits de para OS X. Como resultado, la ejecución de uno de estos comandos desde esta aplicación genera un error 33, "Comando o función no implementado".*

Para reemplazar la tecnología XSLT en sus bases de datos, 4D ofrece dos soluciones:

- *Utilizar las funciones equivalentes del módulo PHP libxslt,que está instalado en 4D partir de la versión 14.2. 4D ofrece un documento específico para ayudarle a utilizar el XSL de PHP para reemplazar comandos XSLT de 4D: Descargue el documento "XSLT con PHP" (PDF)*
- *Utilizar los medios previstos por el comando* ***PROCESS 4D TAGS****, cuyas capacidades se han ampliado considerablemente a partir de 4D v14 R4.*

XML DECODE

XML DECODE (valor XML ; objeto4D)

Parámetro	Tipo		Descripción
valor XML	Texto	➡	Valor de tipo texto proveniente de una estructura XML
objeto4D	Campo, Variable	⬅	Variable o campo 4D que recibe el valor XML convertido

Descripción

*El comando **XML DECODE** convierte un valor guardado como una cadena XML en un valor 4D. La conversión se efectúa automáticamente en función de las siguientes reglas:*

Valor	Ejemplos	Conversión en sistema inglés
numérico	<Price>8,5</Price> <Price>8.5</Price>	Real: 8.5
Booleano	<Double>1</Double> <Double>0</Double> o <Double>true</Double> <Double>false</Double>	Booleano: True/False
BLOB		Decodificación Base64
Imágenes		Decodificación Base64 + comando BLOB to picture
Fechas	2009-10-25T01:03:20+01:00	!10/25/2009! -> Supresión de la parte hora y de la zona horaria
Horas	2009-10-25T01:03:20+01:00	?01:03:20? -> Supresión de la parte fecha. Atención: la zona horaria se tiene en cuenta si es diferente de la hora local. Por ejemplo: "2009-10-25T01:03:20+05:00" se decodificará ?21:03:20? en UTC+1 hora local

Ejemplo

Importación de datos desde un documento XML en el cual los valores se guardan como atributos.
Ejemplo del documento XML:

```
<CD Date="2003-01-01T00:00:00Z" Description="This double CD reissued by EMI in 1995 combines 4 Stabat mater hymns. One by Rossini interpreted by the Berlin Symphony Orchestra, directed by Karl Forster. Followed by a work of Verdi, by the Philharmonic Orchestra, directed by Carlo Maria Giulini. On the second CD, you will find Francis Poulenc interpreted by Régine Crespin. This compilation ends with a little-known version, that of the Polish composer Karol Szymanowski. Polish National Radio Symphony Orchestra directed by Antoni Wit" Double="true" Duration="7246" Type="Sacred music" CD_ID="5" Performer="Various" Price="8.5" Title="4 Stabat mater"/>
```

```
Repeat
   MyEvent:=SAX Get XML node(DocRef)

   Case of
      :(MyEvent=XML Start Element)
         ARRAY TEXT(arrAttrNames;0)
         ARRAY TEXT(arrAttrValues;0)
         SAX GET XML ELEMENT(DocRef;vName;vPrefix;arrAttrNames;arrAttrValues)
         If(vName="CD")
            CREATE RECORD([CD])
            For($i;1;Size of array(arrAttrNames))
               $attrName:=arrAttrNames{$i}
               Case of
                  :($attrName="CD_ID")
                     XML DECODE(arrAttrValues{$i};[CD]CD_ID)
                  :($attrName="Title")
                     [CD]Work:=arrAttrValues{$i}
                  :($attrName="Price")
                     XML DECODE(arrAttrValues{$i};[CD]Price)
                  :($attrName="Date")
                     XML DECODE(arrAttrValues{$i};[CD]Date entered)
                  :($attrName="Duration")
                     XML DECODE(arrAttrValues{$i};[CD]Total_duration)
                  :($attrName="Double")
                     XML DECODE(arrAttrValues{$i};[CD]Double_CD)
               End case
            End for
         End if
```

```
            ...
        End case
    Until (MyEvent=XML End Document)
```

XML GET ERROR

XML GET ERROR (elementRef ; textoError {; linea {; columna}})

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
textoError	Variable	⬅	Texto del error
linea	Variable	⬅	Número de línea
columna	Variable	⬅	Número de columna

Descripción

*El comando **XML GET ERROR** devuelve en el parámetro textoError la descripción del error encontrado durante el proceso del elemento XML designado por el parámetro refElement. La información devuelta es suministrada por la librería Xerces.DLL.*

Los parámetros opcionales linea y columna indican la ubicación del error: ellos recuperan respectivamente el número de línea y en esta línea, la posición del primer carácter de la expresión en el origen del error.

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

XML GET OPTIONS

XML GET OPTIONS (refElement | document ; selector ; valor {; selector2 ; valor2 ; ... ; selectorN ; valorN})

Parámetro	Tipo		Descripción
refElement | document	Texto	➡	Referencia del elemento XML raíz o referenica del documento abierto
selector	Entero largo	➡	Opción a leer
valor	Entero largo	⬅	Valor actual de la opción

Descripción

*El comando **XML GET OPTIONS** se utiliza para obtener el valor actual de uno o más de los parámetros XML definidos para la sesión actual y el usuario actual.*

*En selector, pase una de las constantes del tema **XML** indicando la opción a obtener. El valor actual de la opción se devuelve en el parámetro valor:*

Constante	Tipo	Valor	Comentario
XML binary encoding	*Entero largo*	*5*	*Especifica la manera como se convierten los datos binarios.* ***Valores posibles:*** • *XML Base64 (valor por defecto): los datos binarios se convierten simplemente en base64* • *XML Data URI scheme: los datos binarios se convierten en base64 y se añade el encabezado "data:;base64". Este formato permite principalmente a un navegador decodificar automáticamente una imagen, y también es necesario para insertar imágenes . Para mayor información, consulte http://en.wikipedia.org/wiki/Data_URI_scheme.*
XML date encoding	*Entero largo*	*2*	*Especifica la forma en que se convierten las fechas 4D. Por ejemplo, !01/01/2003! en la zona horaria de Paris.* ***Valores posibles:*** • *XML ISO (valor por defecto): uso del formato xs:datetime sin indicación de la zona horaria. Resultado: "2003-01-01". La parte hora, si está presente en el valor 4D (vía SQL) se pierde.* • *XML Local: uso del formato xs:date con indicación de zona horaria. Resultado: "2003-01-01 +01:00". La parte hora, si está presente en el valor 4D (vía SQL) se pierde.* • *XML Datetime local: uso del formato xs:dateTime (ISO 8601). Indicación de la zona horaria. Este formato permite conservar la parte hora, si está presente en el valor 4D (vía SQL). Resultado: "<Date>2003-01-01T00:00:00 +01:00</Date>".* • *XML UTC: uso del formato xs:date. Resultado: "2003-01-01Z". La parte hora, si está presente en el valor 4D (vía SQL) se pierde.* • *XML Datetime UTC: uso del formato xs:dateTime (ISO 8601). Este formato permite conservar la parte hora, si está presente en el valor 4D (vía SQL). Resultado: "<Date>2003-01-01T00:00:00Z</Date>".*
XML DOM case sensitivity	*Entero largo*	*8*	*Especifica la sensibilidad a mayúsculas y minúsculas con respecto a los nombres de los elementos de los comandos **DOM Get XML element** y **DOM Count XML elements**.* *Valores posibles:* • *XML case sensitive (valor predeterminado): los comandos distinguen entre mayúsculas y minúsculas* • *XML case insensitive: los comandos no distinguen entre mayúsculas y minúsculas.*
XML external entity resolution	*Entero largo*	*7*	*Controla si las entidades externas están definidas en documentos XML. Por razones de seguridad, por defecto, los analizadores XML DOM y SAX de 4D no permiten la resolución de entidades externas. Tenga en cuenta que el alcance de este selector es el proceso de llamada (si es apropiativo) o todos los procesos cooperativos (si se llama desde un proceso cooperativo). Se aplica globalmente a todos los documentos XML (el primer parámetro se ignora, puede pasar una cadena vacía).* ***Valores posibles:*** • *XML enabled: permite la resolución de entidades externas en documentos XML* • *XML disabled (valor predeterminado): no permite la resolución de entidades externas (una declaración de entidad externa genera un error de análisis)*
XML indentation	*Entero largo*	*4*	*Define la indentación del document XML.* ***Valores posibles:*** • *XML With indentation (valor por defecto): el documento está indentado.* • *XML No indentation: el documento no está indentado; su contenido se ubica en una sola línea.*
XML picture encoding	*Entero largo*	*6*	*Especifica la forma en la que las imágenes deben convertirse (antes de codificar en base64).* ***Valores posibles:*** • *XML Convert to PNG (valor por defecto): las imágenes se convierten en PNG antes de ser codificadas en base64.* • *XML Native codec: las imágenes se convierten en su primer CODEC nativo de almacenamiento antes de ser codificadas en base64. Debe utilizar estas opciones para codificar imágenes SVG (ver ejemplo del comando **XML SET OPTIONS**).*
XML string encoding	*Entero largo*	*1*	*Especifica la forma como las cadenas 4D se convierten en valores de elementos. No concierne a las conversiones en atributos para las cuales XML impone el uso de caracteres de escape.* ***Valores posibles:*** • *XML With escaping (valor por defecto): conversión de las cadenas 4D en valores de elementos XML con reemplazo de caracteres. Los datos de tipo texto son analizados automáticamente de manera que los caracteres prohibidos (<&>') son reemplazados por las entidades XML (&<> '").* • *XML Raw data: las cadenas 4D se envían como datos brutos; 4D no efectúa codificación ni análisis. Los valores 4D se convierten si es posible en fragmentos XML y se insertan como hijo del elemento objetivo. Si un valor no puede considerarse como fragmento XML, se inserta en forma de dato bruto en un nuevo nodo CDATA.*

Constante	Tipo	Valor	Comentario
XML time encoding	*Entero largo*	*3*	*Define la forma como las horas 4D se convierten. Por ejemplo, ?02/00/46? (hora de Paris). La codificación difiere dependiendo de si quiere expresar una hora o una duración.* ***Valores posibles para las horas:*** • *XML Datetime UTC: hora expresada en UTC (Universal Time Coordinated). Note que la conversión a UTC es automática. Resultado: "<Duration>0000-00-00T01:00:46Z</Duration>".* • *XML Datetime local: hora expresada con la diferencia horaria de la máquina del motor de 4D. Resultado: "<Duration>0000-00-00T02:00:46+01:00</Duration>".* • *XML Datetime local absolute (valor por defecto): hora expresada sin indicación de la zona horaria. Sin modificación del valor. Resultado: "<Duration>0000-00-00T02:00:46</Duration>".* ***Valores posibles para las duraciones:*** • *XML Seconds: número de segundos desde la media noche; sin modificación del valor porque expresa una duración. Resultado: "<Duration>7246</Duration>".* • *XML Duration: duración expresada conforme a XML Schema Part 2: Datatypes Second Edition. Sin modificación dle valor ya que expresa una duración. Resultado: "<Duration>PT02H00M46S</Duration>".*

XML SET OPTIONS

XML SET OPTIONS (refElement | document ; selector ; valor {; selector2 ; valor2 ; ... ; selectorN ; valorN})

Parámetro	Tipo		Descripción
refElement | document	Texto	➡	Referencia del elemento XML raíz o Referencia del documento abierto
selector	Entero largo	➡	Opción a definir
valor	Entero largo	➡	Valor de la opción

Descripción

El comando XML SET OPTIONS se utiliza para modificar el valor de uno o más parámetros XML para estructura pasada en el primer parámetro.

Este comando se aplica a las estructuras XML de tipo "árbol" (DOM) o "document" (SAX). En el primer parámetro, puede pasar una referencia del elemento raíz (refElement), o la referencia de un documento SAX abierto (document).

*Pase la opción a modificar en selector y el nuevo valor de la opción en valor. Puede pasar tantos pares selector/valor como desee. Debe utilizar las constantes descritas a continuación, colocadas en el tema "**XML**":*

Las siguientes opciones sólo se utilizan en la dirección 4D a XML (no tienen ningún efecto en la lectura de valores XML en 4D) por estos comandos

- **DOM SET XML ATTRIBUTE**
- **DOM SET XML ELEMENT VALUE**
- **SAX ADD XML ELEMENT VALUE**

Constante	Tipo	Valor	Comentario
XML binary encoding	*Entero largo*	*5*	*Especifica la manera como se convierten los datos binarios.* ***Valores posibles:*** • *XML Base64 (valor por defecto): los datos binarios se convierten simplemente en base64* • *XML Data URI scheme: los datos binarios se convierten en base64 y se añade el encabezado "data:;base64". Este formato permite principalmente a un navegador decodificar automáticamente una imagen, y también es necesario para insertar imágenes . Para mayor información, consulte http://en.wikipedia.org/wiki/Data_URI_scheme.*
XML date encoding	*Entero largo*	*2*	*Especifica la forma en que se convierten las fechas 4D. Por ejemplo, !01/01/2003! en la zona horaria de Paris.* ***Valores posibles:*** • *XML ISO (valor por defecto): uso del formato xs:datetime sin indicación de la zona horaria. Resultado: "2003-01-01". La parte hora, si está presente en el valor 4D (vía SQL) se pierde.* • *XML Local: uso del formato xs:date con indicación de zona horaria. Resultado: "2003-01-01 +01:00". La parte hora, si está presente en el valor 4D (vía SQL) se pierde.* • *XML Datetime local: uso del formato xs:dateTime (ISO 8601). Indicación de la zona horaria. Este formato permite conservar la parte hora, si está presente en el valor 4D (vía SQL). Resultado: "<Date>2003-01-01T00:00:00 +01:00</Date>".* • *XML UTC: uso del formato xs:date. Resultado: "2003-01-01Z". La parte hora, si está presente en el valor 4D (vía SQL) se pierde.* • *XML Datetime UTC: uso del formato xs:dateTime (ISO 8601). Este formato permite conservar la parte hora, si está presente en el valor 4D (vía SQL). Resultado: "<Date>2003-01-01T00:00:00Z</Date>".*
XML indentation	*Entero largo*	*4*	*Define la indentación del document XML.* ***Valores posibles:*** • *XML With indentation (valor por defecto): el documento está indentado.* • *XML No indentation: el documento no está indentado; su contenido se ubica en una sola línea.*
XML picture encoding	*Entero largo*	*6*	*Especifica la forma en la que las imágenes deben convertirse (antes de codificar en base64).* ***Valores posibles:*** • *XML Convert to PNG (valor por defecto): las imágenes se convierten en PNG antes de ser codificadas en base64.* • *XML Native codec: las imágenes se convierten en su primer CODEC nativo de almacenamiento antes de ser codificadas en base64. Debe utilizar estas opciones para codificar imágenes SVG (ver ejemplo del comando **XML SET OPTIONS**).*
XML string encoding	*Entero largo*	*1*	*Especifica la forma como las cadenas 4D se convierten en valores de elementos. No concierne a las conversiones en atributos para las cuales XML impone el uso de caracteres de escape.* ***Valores posibles:*** • *XML With escaping (valor por defecto): conversión de las cadenas 4D en valores de elementos XML con reemplazo de caracteres. Los datos de tipo texto son analizados automáticamente de manera que los caracteres prohibidos (<&>') son reemplazados por las entidades XML (&<> '").* • *XML Raw data: las cadenas 4D se envían como datos brutos; 4D no efectúa codificación ni análisis. Los valores 4D se convierten si es posible en fragmentos XML y se insertan como hijo del elemento objetivo. Si un valor no puede considerarse como fragmento XML, se inserta en forma de dato bruto en un nuevo nodo CDATA.*
XML time encoding	*Entero largo*	*3*	*Define la forma como las horas 4D se convierten. Por ejemplo, ?02/00/46? (hora de Paris). La codificación difiere dependiendo de si quiere expresar una hora o una duración.* ***Valores posibles para las horas:*** • *XML Datetime UTC: hora expresada en UTC (Universal Time Coordinated). Note que la conversión a UTC es automática. Resultado: "<Duration>0000-00-00T01:00:46Z</Duration>".* • *XML Datetime local: hora expresada con la diferencia horaria de la máquina del motor de 4D. Resultado: "<Duration>0000-00-00T02:00:46+01:00</Duration>".* • *XML Datetime local absolute (valor por defecto): hora expresada sin indicación de la zona horaria. Sin modificación del valor. Resultado: "<Duration>0000-00-00T02:00:46</Duration>".* ***Valores posibles para las duraciones:*** • *XML Seconds: número de segundos desde la media noche; sin modificación del valor porque expresa una duración. Resultado: "<Duration>7246</Duration>".* • *XML Duration: duración expresada conforme a XML Schema Part 2: Datatypes Second Edition. Sin modificación dle valor ya que expresa una duración. Resultado: "<Duration>PT02H00M46S</Duration>".*

Notas:

- *Los valores XML Local y XML Datetime local no ofrecen fechas expresadas en UTC (Universal Time Coordinated); se convierten sin modificación pero indican la diferencia horaria. Estos formatos son útiles en el caso de conversiones sucesivas y recíprocas (round tripping).*

- *Los valores XML UTC y XML Datetime UTC son equivalentes a los precedentes desde el punto de vista del formato, pero se expresan en UTC. Estos formatos deben tener prioridad para asegurar la interoperabilidad. Los valores no son modificables.*

Las siguientes opciones le permiten modificar algunas funcionalidades del analizador xml predeterminado:

Constante	Tipo	Valor	Comentario
XML DOM case sensitivity	*Entero largo*	*8*	*Especifica la sensibilidad a mayúsculas y minúsculas con respecto a los nombres de los elementos de los comandos **DOM Get XML element** y **DOM Count XML elements**. Valores posibles:* • *XML case sensitive (valor predeterminado): los comandos distinguen entre mayúsculas y minúsculas* • *XML case insensitive: los comandos no distinguen entre mayúsculas y minúsculas.*
XML external entity resolution	*Entero largo*	*7*	*Controla si las entidades externas están definidas en documentos XML. Por razones de seguridad, por defecto, los analizadores XML DOM y SAX de 4D no permiten la resolución de entidades externas. Tenga en cuenta que el alcance de este selector es el proceso de llamada (si es apropiativo) o todos los procesos cooperativos (si se llama desde un proceso cooperativo). Se aplica globalmente a todos los documentos XML (el primer parámetro se ignora, puede pasar una cadena vacía).* ***Valores posibles:*** • *XML enabled: permite la resolución de entidades externas en documentos XML* • *XML disabled (valor predeterminado): no permite la resolución de entidades externas (una declaración de entidad externa genera un error de análisis)*

Ejemplo

Inserción de una imagen SVG:

```
XML SET OPTIONS(pictElemRef;XML binary encoding;XML data URI scheme)
XML SET OPTIONS(pictElemRef;XML picture encoding;XML native codec)
DOM SET XML ATTRIBUTE(pictElemRef;"xlink:href";PictVar)
```

_o_XSLT APPLY TRANSFORMATION

_o_XSLT APPLY TRANSFORMATION (fuenteXML ; hojaXSL ; resultado {; compileHoja})

Parámetro	Tipo		Descripción
fuenteXML	Cadena, BLOB	➡	Nombre o ruta de acceso del documento XML fuente o BLOB que contiene el XML fuente
hojaXSL	Cadena, BLOB	➡	Nombre o ruta de acceso del documento que contiene la hoja de estilo XSL o BLOB que contiene la hoja de estilo XSL
resultado	Cadena, BLOB	➡	Nombre o ruta de acceso del documento que recibe el resultado de la transformación XSLT o BLOB que recibe el resultado de la transformación XSLT
compileHoja	Booleano	➡	True = Optimiza la transformación XSLT False o si se omite = No optimización, borrar el archivo XSL compilado (si lo hay)

Compatibility note

A partir de 4D v14 R4, los comandos de transformación XSL son obsoletos. Por razones de compatibilidad, aún están soportados en 4D pero le recomendamos que deje de utilizarlos. En futuras versiones de 4D, ya no será posible utilizar la tecnología XSLT. Para obtener más información, consulte la sección ***Presentación de los comandos XML utilitarios*** *.*

_o_XSLT GET ERROR

_o_XSLT GET ERROR (textoError {; linea {; columna}})

Parámetro	Tipo		Descripción
textoError	*Variable*	⇐	*Texto del error*
linea	*Variable*	⇐	*Número de la línea*
columna	*Variable*	⇐	*Número de la columna*

Compatibility note

A partir de 4D v14 R4, los comandos de transformación XSL son obsoletos. Por razones de compatibilidad, aún están soportados en 4D pero le recomendamos que deje de utilizarlos. En futuras versiones de 4D, ya no será posible utilizar la tecnología XSLT. Para obtener más información, consulte la sección ***Presentación de los comandos XML utilitarios*** *.*

_o_XSLT SET PARAMETER

_o_XSLT SET PARAMETER (nomParam ; valorParam)

Parámetro	Tipo		Descripción
nomParam	Cadena	➡	Nombre del parámetro a buscar en la hoja XSL
valorParam	Cadena	➡	Valor del parámetro a utilizar en el documento transformado

Compatibility note

A partir de 4D v14 R4, los comandos de transformación XSL son obsoletos. Por razones de compatibilidad, aún están soportados en 4D pero le recomendamos que deje de utilizarlos. En futuras versiones de 4D, ya no será posible utilizar la tecnología XSLT. Para obtener más información, consulte la sección ***Presentación de los comandos XML utilitarios*** *.*

XML DOM

- *Presentación de los comandos XML DOM*
- *DOM Append XML child node*
- *DOM Append XML element*
- *DOM CLOSE XML*
- *DOM Count XML attributes*
- *DOM Count XML elements*
- *DOM Create XML element*
- *DOM Create XML element arrays*
- *DOM Create XML Ref*
- *DOM EXPORT TO FILE*
- *DOM EXPORT TO VAR*
- *DOM Find XML element*
- *DOM Find XML element by ID*
- *DOM Get first child XML element*
- *DOM Get last child XML element*
- *DOM Get next sibling XML element*
- *DOM Get parent XML element*
- *DOM Get previous sibling XML element*
- *DOM Get Root XML element*
- *DOM GET XML ATTRIBUTE BY INDEX*
- *DOM GET XML ATTRIBUTE BY NAME*
- *DOM GET XML CHILD NODES*
- *DOM Get XML document ref*
- *DOM Get XML element*
- *DOM GET XML ELEMENT NAME*
- *DOM GET XML ELEMENT VALUE*
- *DOM Get XML information*
- *DOM Insert XML element*
- *DOM Parse XML source*
- *DOM Parse XML variable*
- *DOM REMOVE XML ATTRIBUTE*
- *DOM REMOVE XML ELEMENT*
- *DOM SET XML ATTRIBUTE*
- *DOM SET XML DECLARATION*
- *DOM SET XML ELEMENT NAME*
- *DOM SET XML ELEMENT VALUE*

Presentación de los comandos XML DOM

4D incluye un conjunto de comandos utilizados para analizar objetos que contienen los datos XML (eXtensible Markup Language). ***Nota relativa al modo apropiativo****: las referencias XML creadas por un proceso apropiativo sólo se puede utilizar en este proceso específico. Por el contrario, las referencias XML creadas por un proceso cooperativo pueden ser utilizadas por cualquier otro proceso cooperativo, pero no pueden ser utilizadas por cualquier proceso apropiativo.*

A propósito del lenguaje XML

El lenguaje XML es un estándar de intercambio de datos. Está basado en el uso de etiquetas y permite describir de manera precisa los datos intercambiados así como su estructura. Los archivos XML son archivos en formato Texto, su contenido es analizado (parsing) por las aplicaciones que importan los datos. Hoy en día, muchas aplicaciones soportan este formato.

Para mayor información sobre XML, consulte, por ejemplo, los sitios http://xml.org y http://www.w3.org.

Para soportar XML, 4D utiliza una librería llamada Xerces.dll desarrollada por la sociedad Apache Foundation. 4D soporta XML versión 1.0.

Nota*: 4D permite importar y exportar directamente los datos en formato XML utilizando el editor de importación/exportación.*

Comandos DOM y SAX

Los comandos de este tema tienen el prefijo DOM. De hecho, 4D ofrece dos conjuntos independientes de comandos XML, con prefijo DOM y SAX: DOM (Document Object Model) y SAX (Simple API XML) son dos modos de análisis diferentes de los documentos XML.

• El modo DOM analiza una fuente XML y crea su estructura (su "árbol") en memoria. Por esta razón, el acceso a cada elemento de la fuente es extremadamente rápido. Sin embargo, como la totalidad del árbol se almacena en memoria, el procesamiento de documentos XML grandes podría exceder la capacidad de memoria y por lo tanto provocar errores.
*• El modo SAX no crea un árbol en memoria. En este modo, los "eventos" (tales como el inicio o el fin de un elemento) son generados durante el análisis de la fuente. Este modo le permite analizar documentos XML de cualquier tamaño, sin importar la cantidad de memoria disponible. Los comandos SAX se agrupan en el tema "****XML SAX****". Para mayor información, consulte la sección* ***Presentación de los comandos XML SAX****.*
Para mayor información sobre estándares XML, consulte los siguientes sitios: http://www.saxproject.org/?selected=event y http://www.w3schools.com/xml/.

Creación, apertura y cierre de documentos XML vía DOM

Los objetos creados, modificados o analizados por los comandos DOM de 4D pueden ser de tipo texto, URLs, documentos o BLOBs. Los comandos DOM utilizados para la apertura de los objetos XML en 4D son ***Parse XML source*** *y* ***DOM Parse XML variable****.*
Varios comandos le permiten leer, analizar y escribir los elementos y los atributos. La recuperación de errores se efectúa vía el comando ***XML GET ERROR*** *(común para ambos estándares XML).*
El comando ***DOM CLOSE XML*** *permite cerrar la fuente al final.*

Nota sobre el uso de parámetros BLOB XML: *las estructuras XML están basadas en datos de tipo texto, se recomienda utilizar variables o campos de tipo texto para manipularlas. Por razones históricas, los comandos XML de 4D (por ejemplo* ***DOM Parse XML variable****) aceptan parámetros de tipo BLOB. En versiones anteriores de 4D, el tamaño de las variables de tipo Texto está limitado a 32 KB. A partir de la versión 11 de 4D, las variables y los campos de tipo texto pueden contener hasta 2 GB de datos. Como se eliminó la limitación original, ahora no se recomienda almacenar texto en BLOBs. El uso de BLOBs está reservado para el procesamiento de datos binarios. Conforme con las especificaciones XML, a partir de 4D v12, los datos binarios son codificados automáticamente en Base64, incluso cuando el BLOB contiene texto.*

Uso de la notación XPath

*Tres comandos XML DOM (****DOM Create XML element****,* ***DOM Find XML element*** *y* ***DOM SET XML ELEMENT VALUE)*** *aceptan la notación XPath para el acceso a los elementos XML.*

La notación XPath viene del lenguaje XPath, designado a la navegación dentro de estructuras XML. Permite designar directamente los elementos dentro de una estructura XML vía una sintaxis de tipo "ruta de acceso", sin tener que indicar necesariamente la ruta completa. Por ejemplo, dada la siguiente estructura:

```
<RootElement>  <Elem1>  <Elem2>  <Elem3 Font=Verdana Size=10> </Elem3>  </Elem2>
  </Elem1>  </RootElement>
```

La notación XPath permite acceder al elemento 3 utilizando la sintaxis /RootElement/Elem1/Elem2/Elem3.

4D acepta igualmente los elementos XPath ***indexados****, utilizando la sintaxis* ***Element[ElementNum]****. Por ejemplo, dada la siguiente estructura:*

```
<RootElement>  <Elem1>  <Elem2>aaa</Elem2>  <Elem2>bbb</Elem2>  <Elem2>ccc</Elem2>
  </Elem1>  </RootElement>
```

La notación XPath permite acceder al valor "ccc" utilizando la sintaxis /RootElement/Elem1/Elem2[3].

*Para una ilustración de la notación XPath, por favor consulte los ejemplos de los comandos **DOM Create XML element** y **DOM Find XML element**.*

Conjuntos de caracteres

Los siguientes conjuntos de caracteres son soportados por los comandos XML DOM y XML SAX de 4D:

- *ASCII*
- *UTF-8*
- *UTF-16 (Big/Small Endian)*
- *UCS4 (Big/Small Endian)*
- *EBCDIC code pages IBM037, IBM1047 y IBM1140 encodings,*
- *ISO-8859-1 (o Latin1)*
- *Windows-1252.*

Terminología

El lenguaje XML utiliza numerosos términos y acrónimos específicos. Esta lista no exhaustiva detalla los principales conceptos XML utilizados por los comandos y funciones de 4D.

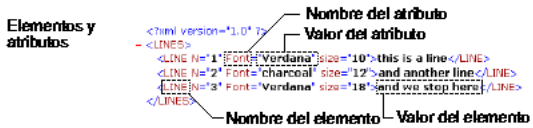
***Atributo:** una subetiqueta XML asociada a un elemento. Un atributo siempre contiene un nombre y un valor (ver diagrama a continuación).*

***Hijo**: en una estructura XML, un elemento en un nivel directamente inferior a otro.*

***DTD**: Document Tipo Declaration (Declaración de tipo de documento). La DTD graba el conjunto de reglas y de propiedades específicas que debe seguir un documento XML. Estas reglas definen, más particularmente, el nombre y contenido de cada etiqueta como también su contenido. Esta formalización de los elementos permite verificar que un documento XML esté en conformidad (en ese caso, se declara "válido").*

La DTD puede incluirse en el documento XML (DTD interna) o en un documento separado (DTD externo). Note que la DTD no es obligatoria.

***Elemento**: una etiqueta XML. Un elemento siempre contiene un nombre y un valor. Opcionalmente, un elemento puede contener atributos (ver diagrama).*



***RefElement**: referencia XML utilizada por los comandos XML de 4D para especificar una estructura XML. Esta referencia está formada por 8 caracteres codificados en forma hexadécimal, lo cual significa que su longitud es de 16 o 32 caracteres dependiendo de si utiliza un sistema de 32 o 64 bits. Se recomienda declarar las referencias XML utilizando la directiva **C_TEXT**.*

***Padre**: en una estructura XML, un elemento de un nivel directamente superior a otro.*

***Parsing**, **parser (Analizador)**: acción de analizar el contenido de un objeto estructurado para extraer información útil. Los comandos del tema "XML" se utilizan para analizar el contenido de todo objeto XML.*

***Raíz (Root)**: elemento ubicado en el primer nivel de una estructura XML.*

***Hermano**: en una estructura XML, elemento del mismo nivel que otro.*

***Estructura XML**: objeto XML estructurado. Este objeto puede ser un documento, una variable, o un elemento.*

***Validación**: un documento XML es "validado" por el analizador XML cuando está "bien formado" y conforme con las especificaciones de la DTD. Ver también Bien formado.*

***Bien formado**: un documento XML es declarado "bien formado" por el analizador XML cuando cumple con las especificaciones XML genéricas. Ver también Validación.*

***XML**: eXtensible Markup Language (Lenguaje de etiquetas evolutivo). Estándar de intercambio de datos computarizado que permite transferir datos como también su estructura. El lenguaje XML está basado en el uso de etiquetas y una sintaxis específica, de acuerdo con el lenguaje HTML. Sin embargo, a diferencia de éste último, el lenguaje XML permite la definición de etiquetas personalizadas.*

***XSL**: eXtensible Stylesheet Language (Lenguaje de hojas de estilo evolutivo). Lenguaje que permite definir las hojas de estilo utilizadas para procesar y visualizar los contenidos de un documento XSL.*

Gestión de errores

Varias funciones de este tema devuelven una referencia de elemento XML. Si ocurre un error durante la ejecución de una función, por ejemplo si la referencia del elemento raíz es inválida, la variable OK toma el valor 0 y el comando genera un error.

Además, la referencia devuelta en este caso es una secuencia de caracteres "0" (16 caracteres en 32 bits, o 32 caracteres en 64 bits).

DOM Append XML child node

DOM Append XML child node (refElement ; tipoHijo ; valorHijo) -> Resultado

Parámetro	Tipo		Descripción
refElement	Texto	➡	Referencia del elemento XML
tipoHijo	Entero largo	➡	Tipo de hijo a añadir
valorHijo	Texto, BLOB	➡	Texto o variable (Texto o BLOB)cuyo valor debe insertarse como nodo hijo
Resultado	Texto	⮌	Referencia del elemento XML hijo

Descripción

*El comando **DOM Append XML child node** se utiliza para anexar el valor valorHijo al nodo XML designado por refElement.*

El tipo de nodo creado es especificado por el parámetro tipoHijo. En este parámetro se puede pasar una de las siguientes constantes, del tema "":

Constante	Tipo	Valor
XML CDATA	Entero largo	7
XML comment	Entero largo	2
XML DATA	Entero largo	6
XML DOCTYPE	Entero largo	10
XML ELEMENT	Entero largo	11
XML processing instruction	Entero largo	3

En valorHijo, pase los datos a insertar. Pase una cadena o una variable 4D (cadena o BLOB). El contenido de este parámetro siempre se convertirá en texto.

***Nota**: si el parámetro refElement designa el nodo Document (nodo de nivel superior), el comando inserta un nodo "Doctype" antes de cualquier otro nodo. Lo mismo ocurre con las instrucciones de procesamiento y los comentarios, que siempre se insertan antes del nodo raíz (pero después del nodo Doctype).*

Ejemplo 1

Adición de un nodo de tipo texto:

```
Reference:=DOM Create XML element(refElement;"myElement")
DOM SET XML ELEMENT VALUE(Reference;"Hola")
temp:=DOM Create XML element(Reference;"br")
temp:=DOM Append XML child node(Reference;XML DATA;"Nueva")
temp:=DOM Create XML element(Reference;"br")
temp:=DOM Append XML child node(Reference;XML DATA;"York")
```

Resultado:

```
<myElement>Hola<br/>Nueva<br/>York</myElement>
```

Ejemplo 2

Adición de un nodo de tipo instrucción de proceso:

```
$Txt_instruction:="xml-stylesheet type = \"text/xsl\" href=\"style.xsl\""
Reference:=DOM Append XML child node(elementRef;XML Processing Instruction;$Txt_instruction)
```

Resultado (insertado antes del primer elemento):

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

Ejemplo 3

Adición de un nodo de tipo comentario:

```
Reference:=DOM Append XML child node(elementRef;XML Comment;"Hola mundo")
```

Resultado:

```
<!--Hola mundo-->
```

Ejemplo 4

Adición de un nodo de tipo CDATA:

```
Reference:=DOM Append XML child node(elementRef;XML CDATA;"12 < 18")
```

Resultado:

```
<element><![CDATA[12 < 18]]></element>
```

Ejemplo 5

Adición o reemplazo de un nodo de tipo declaración Doctype:

```
Reference:=DOM Append XML child node(elementRef;XML DOCTYPE;"Books SYSTEM \"Book.DTD\"")
```

Resultado (insertado antes del primer elemento):

```
<!DOCTYPE Books SYSTEM "Book.DTD">
```

Ejemplo 6

Adición o reemplazo de un nodo de tipo Element.

- *si el parámetro valorHijo es un fragmento XML, se inserta como nodos hijos:*

  ```
  Reference:=DOM Append XML child node(refElement;XML ELEMENT;"<child>simon</child><child>eva</child>")
  ```

  Resultado:

  ```
  <parent>  <child>simon</child>  <child>eva</child> </parent>
  ```

- *de lo contrario, se añade un nuevo elemento hijo vacío:*

  ```
  Reference:=DOM Append XML child node(elementRef;XML ELEMENT;"tbreak")
  ```

  Resultado:

  ```
  <parent>  <tbreak/> </parent>
  ```

Si el contenido de valorHijo no es valido, se devuelve un error.

DOM Append XML element

DOM Append XML element (refElementTarget ; refElementFuente) -> Resultado

Parámetro	Tipo		Descripción
refElementTarget	Texto	→	Referencia del elemento XML padre
refElementFuente	Texto	→	Referencia del elemento XML a añadir
Resultado	Texto	←	Referencia del nuevo elemento XML

Descripción

*El comando **DOM Append XML element** se utiliza para añadir un nuevo elemento XML al hijo del elemento XML cuya referencia se pasa en el parámetro refElementFuente .*

En el parámetro refElementFuente, pase el elemento a añadir. Este elemento debe pasarse como referencia de un elemento XML existente en un árbol DOM. Se añade luego del último elemento hijo existente de refElementTarget.

Ejemplo

*Ver el ejemplo del comando **DOM Insert XML element**.*

DOM CLOSE XML

DOM CLOSE XML (elementRef)

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML raíz

Descripción

*El comando **DOM CLOSE XML** libera el espacio en memoria ocupado por el objeto XML designado por refElement.*
Si refElement no es un objeto XML raíz, se genera un error.

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM Count XML attributes

DOM Count XML attributes (elementRef) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
Resultado	Entero largo	⮌	Número de atributos

Descripción

*El comando **DOM Count XML attributes** devuelve el número de los atributos XML presentes en el elemento XML designado por refElement. Para mayor información sobre los atributos XML, consulte la sección **Presentación de los comandos XML DOM**.*

Ejemplo

Antes de recuperar los valores de los elementos en un array, usted quiere conocer el número de atributos en el siguiente elemento:

```
<?xml version="1.0"?>
<LINES>
    <LINE N="1"Font="Verdana" size="10">esta es una línea</LINE>
    <LINE N="1"Font="charcoal" size="10">y otra línea</LINE>
    <LINE N="1"Font="Verdana" size="10">y nos detenemos aquí</LINE>
</LINES>
```

```
C_BLOB(myBlobVar)
C_TEXT($xml_Parent_Ref;$xml_Child_Ref)
C_TEXT(myResult)
C_LONGINT($numAttributes)

$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get first child XML element($xml_Parent_Ref)

$numAttributes:=DOM Count XML attributes($xml_Child_Ref)
ARRAY TEXT(tAttrib;$numAttributes)
ARRAY TEXT(tValAttrib;$numAttributes)
For($i;1;$numAttributes)
    DOM GET XML ATTRIBUTE BY INDEX($xml_Child_Ref;$i;tAttrib{$i};tValAttrib{$i})
End for
```

En el ejemplo anterior, $numAttributes es igual a 3, tAttrib{1} contiene "Font", tAttrib{2} contiene "N", tAttrib{3} contiene "size" y tValAttrib contiene "Verdana", "1" y "10".
***Nota:** el número de índice no corresponde a la ubicación del atributo en el archivo XML mostrado en forma de texto. En XML, el índice de un atributo indica su posición entre los atributos clasificados por orden alfabético (en función de su nombre).*

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM Count XML elements

DOM Count XML elements (elementRef ; nomElement) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	→	Referencia del elemento XML
nomElement	Cadena	→	Nombre de los elementos XML a contar
Resultado	Entero largo	↩	Número de elementos

Descripción

El comando ***DOM Count XML elements*** *devuelve el número de elementos "hijo" dependientes del elemento padre refElement y llamado nomElement.*

Nota: *por defecto,* ***DOM Count XML elements*** *es sensible a las mayúsculas y minúsculas con respecto el parámetro nomElement (de acuerdo a los estándares XML). Puede controlar la sensibilidad de mayúsculas y minúsculas del comando utilizando el selector XML DOM case sensitivity del comando* ***XML SET OPTIONS****.*

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM Create XML element

DOM Create XML element (elementRef ; xRuta {; nomAtrib ; valorAtrib} {; nomAtrib2 ; valorAtrib2 ; ... ; nomAtribN ; valorAtribN}) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML raíz
xRuta	Texto	➡	Ruta XPath del elemento XML a crear
nomAtrib	Cadena	➡	Atributo a definir
valorAtrib	Cadena, Booleano, Entero largo, Real, Hora, Fecha	➡	Nuevo valor del atributo
Resultado	Cadena	↩	Referencia del elemento XML creado

Descripción

*El comando **DOM Create XML element** permite crear un nuevo elemento en el elemento XML refElement en la ubicación definida por el parámetro xRuta y añadirle atributos si es necesario.*

*Pase en refElement la referencia del elemento raíz (creado, por ejemplo con la ayuda del comando **DOM Create XML Ref**).*

*En xRuta, pase la ruta de acceso del elemento a crear en notación XPath (ver el párrafo "Uso de la notación XPath" en la sección **Presentación de los comandos XML DOM**). Si los elementos de la ruta de acceso no existen, son creados.*

Es posible pasar directamente en xRuta un nombre de elemento simple con el fin de crear un subelemento a partir del elemento actual (ver el ejemplo 3).

***Nota:** si defineo uno o más espacios de nombre (namespaces) para el árbol designado por refElement (ver el comando **DOM Create XML Ref**), debe prefijar el parámetro xRuta del nombre de espacio a utilizar (por ejemplo, "MiNombreEspacio:MiElemento").*

Puede pasar en los parámetros opcionales nomAtributo y valorAtributo un par atributo/valor (en forma de variables, campos o valores literales). Puede pasar tantos pares como quiera.

El parámetro valorAtributo puede ser de tipo texto o de otro tipo (Booleano, entero, real, hora o fecha). Si pasa un valor de un tipo diferente al tipo texto, 4D se encarga de la conversión a texto, de acuerdo a los siguientes principios:

Tipo	Ejemplo de valor convertido
Booleano	"true" o "false" (no traducido)
Entero	"123456"
Real	"12.34" (el separador decimal siempre es ".")
Fecha	"2006-12-04T00:00:00Z" (norma RFC 3339)
Hora	"5233" (Número de segundos)

El comando devuelve en resultado la referencia XML del elemento creado.

Ejemplo 1

Queremos crear el siguiente elemento:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <RootElement>  <Elem1>  <Elem2>  <Elem3> </Elem3>
    <Elem3> </Elem3>  </Elem2>  </Elem1> </RootElement>
```

Para hacerlo, simplemente escribimos:

```
C_TEXT(vRefRaiz;vRefElem)
vRefRaiz:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElem:=DOM Create XML element(vRefRaiz;vxPath)
vxPath:="/RootElement/Elem1/Elem2/Elem3[2]"
vRefElem:=DOM Create XML element(vRefRaiz;vxPath)
```

Ejemplo 2

Queremos crear el siguiente elemento (contiene los atributos):

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <RootElement>  <Elem1>  <Elem2>  <Elem3 Font=Verdana
Size=10> </Elem3>  </Elem2>  </Elem1> </RootElement>
```

Para hacerlo, simplemente escribimos:

```
C_TEXT(vRefRaiz;vRefElem)
C_TEXT($aAtribNom1;$aAtribNom2;$aAtribVal1;$aAtribVal2)
$aAtribNom1:="Font"
$aAtribNom2:="Size"
```

```
$aAtribVal1:="Verdana"
$aAtribVal2:="10"

vRefRaiz:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vRefElem:=DOM Create XML element(vRefRaiz;vxPath;$aAtribNom1;$aAtribVal1;$aAtribNom2;$aAtribVal2)
```

Ejemplo 3

Queremos crear y exportar la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <Root>      <Elem1>Hola</Elem1> </Root>
```

Queremos utilizar la sintaxis basada en un nombre de elemento simple. Para hacerlo, simplemente escribimos:

```
C_TEXT($root)
C_TEXT($ref1)

$root:=DOM Create XML Ref("Raíz")
$ref1:=DOM Create XML element($root;"Elem1")
DOM SET XML ELEMENT VALUE($ref1;"Hola")
DOM EXPORT TO FILE($root;"midoc.xml")
DOM CLOSE XML($root)
```

Variables y conjuntos del sistema

Si el comando se ejecutó correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0 y se genera un error.

Gestión de errores

Se genera un error cuando:

- *La referencia del elemento raíz no es válida.*
- *El nombre del elemento a crear es inválido (por ejemplo, si comienza por un número).*

DOM Create XML element arrays

DOM Create XML element arrays (refElement ; xRuta {; arrayNomsAtrib ; arrayValoresAtrib} {; arrayNomsAtrib2 ; arrayValoresAtrib2 ; ... ; arrayNomsAtribN ; arrayValoresAtribN}) -> Resultado

Parámetro	Tipo		Descripción
refElement	Texto	➡	Referencia del elemento XML raíz
xRuta	Texto	➡	Ruta XRuta del elemento XML a crear
arrayNomsAtrib	Array cadena	➡	Array de nombres de atributos
arrayValoresAtrib	Array cadena	➡	Array de valores de atributos
Resultado	Texto	⮌	Referencia del elemento XML creado

Descripción

*El comando **DOM Create XML element arrays** se utiliza para añadir un nuevo elemento en el elemento XML de refElement, como también opcionalmente, atributos y sus valores en forma de arrays.*

*Aparte de soportar arrays (ver a continuación), este comando es idéntico a **DOM Create XML element**. Por favor consulte la descripción de este comando para conocer mayores detalles de su funcionamiento.*

*Opcionalmente, el comando **DOM Create XML element arrays** permite pasar varios pares de atributos y valores de atributos en forma de arrays en los parámetros arrayNomsAtrib y arrayValoresAtrib. En arrayValoresAtrib, puede pasar arrays de tipo texto, fecha, numérico e imagen. 4D efectúa automáticamente las conversiones necesarias; puede modificar estas conversiones utilizando el comando **XML SET OPTIONS**.*

Los arrays deben haber sido creados previamente y funcionar por pares. Puede pasar tantos pares de arrays y de elementos como quiera en cada par.

Ejemplo

Queremos crear el siguiente elemento:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <RootElement>  <Elem1>  <Elem2>  <Elem3 Font="Verdana"
Size="10" Style="Bold"></Elem3>  </Elem2>  </Elem1> </RootElement>
```

Para esto, es suficiente escribir:

```
ARRAY TEXT(arrAttNames;3)
ARRAY TEXT(arrAttValues;3)
arrAttNames{1}:="Font"
arrAttValues{1}:="Verdana"
arrAttNames{2}:="Size"
arrAttValues{2}:="10"
arrAttNames{3}:="Style"
arrAttValues{3}:="Bold"
vRootRef:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vElementRef:=DOM Create XML element arrays(vRootRef;vxPath;arrAttNames;arrAttValues)
```

DOM Create XML Ref

DOM Create XML Ref (raiz {; nomEspacio} {; nSNom ; nSValor} {; nSNom2 ; nSValor2 ; ... ; nSNomN ; nSValorN}) -> Resultado

Parámetro	Tipo		Descripción
raiz	Cadena	➡	Nombre del elemento raíz
nomEspacio	Cadena	➡	Valor del espacio de nombre (namespace)
nSNom	Cadena	➡	Nombre del espacio de nombre
nSValor	Cadena	➡	Valor de espacio de nombre
Resultado	Cadena	⬅	Referencia del elemento XML raíz

Descripción

*El comando **DOM Create XML Ref** crea un árbol XML vacío en memoria y devuelve su referencia.*

Pase en el parámetro raiz el nombre del elemento raíz del árbol XML.

Pase en el parámetro opcional nomEspacio la declaración del valor del espacio de nombre (namespace) del árbol (por ejemplo "http://www.4dhispano.com").

Note que es posible poner un prefijo al parámetro raiz con el nombre del espacio seguido de dos puntos : (por ejemplo "MiNombreEspacio:MiRaiz"). En este caso, el parámetro nomEspacio especificando el valor del espacio de nombre es obligatorio.

***Nota:** el espacio de nombre es una cadena que permite garantizar la unicidad de los nombres de las variables XML. Por lo general, se utiliza un URL como http://www.misitio.com/miurl. El URL no necesariamente tiene que ser válido, pero tiene que ser único.*

Puede declarar uno o varios espacios de nombre adicionales en el árbol XML generado, con la ayuda de pares nSNombre/nSValor. Puede pasar tantos pares nombre/valor de espacio de nombre como quiera.

***Importante:** recuerde llamar al comando **DOM CLOSE XML** con el fin de liberar memoria cuando termine de utilizar el árbol XML.*

Ejemplo 1

Creación de un árbol XML simple:

```
C_TEXT(vRefElem)
vRefElem:=DOM Create XML Ref("MiRaiz")
```

Este código produce el siguiente resultado:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <MiRaiz/>
```

Ejemplo 2

Creación de un árbol XML con un espacio de nombre:

```
C_TEXT(vRefElem)
$Raiz:="MiNombreEspacio:MiRaiz"
$Nombreespacio:="http://www.4DHispano.com/tech/nombreespacio"
vRefElem:=DOM Create XML Ref($Raiz;$Nombreespacio)
```

Este código da el siguiente resultado:

```
<Minombreespacio:Miraiz xmlns:Minombreespacio="http://www.4DHispano.com/tech/nombreespacio"/>
```

Ejemplo 3

Creación de un árbol XML con varios espacios de nombre:

```
C_TEXT(vRefElem)
C_TEXT(80;$aNSNom1;$aNSNom2;$aNSValor1;$aNSValor2)
$Raiz:="MiNombreEspacio:MiRaiz"
$Nombreespacio:="http://www.4DHispano.com/tech/nombreespacio"
$aNSNom1:="NSNom1"
$aNSNom2:="NSNom2"
$aNSValor1:="http://www.4DHispano.com/Prod/nombreespacio"
```

```
$aNSValor2:="http://www.4DHispano.com/Mkt/nombreespacio"
vRefElem:=DOM Create XML Ref($Raiz;$Nombreespacio;$aNSNom1;$aNSValor1;$aNSNom2;$aNSValor2)
```

Este código produce el siguiente resultado:

```
<MiNombreEspacio:MiRaiz xmlns:MiNombreEspacio="http://www.4DHispano.com/tech/nombreEspacio"
NSNom1="http://www.4DHispano.com/Prod/nombreespacio" NSNom2="http://www.4DHispano.com/Mkt/nombreespacio"/>
```

Variables y conjuntos del sistema

Si el comando se ejecutó correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0 y se genera un error.

DOM EXPORT TO FILE

DOM EXPORT TO FILE (elementRef ; rutaArchivo)

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML raíz
rutaArchivo	Texto	➡	Ruta de acceso completa del archivo

Descripción

El comando ***DOM EXPORT TO FILE*** *permite guardar un árbol XML en un archivo en el disco.*
Pase en elementRef la referencia del elemento raíz a exportar.
Pase en rutaArchivo la ruta de acceso completa del archivo de exportación a utilizar o a crear. Si el archivo no existe, se crea.

Si pasa únicamente un nombre de archivo (sin ruta de acceso), se buscará el archivo o se creará junto al archivo de estructura.
Si pasa una cadena vacía (""), aparece una caja de diálogo estándar de apertura y creación de archivos.

Notas sobre el procesamiento de caracteres de fin de línea

En XML, los saltos de línea no son significativos, independientemente de si se encuentran dentro o entre los elementos XML. Internamente, XML utiliza caracteres estándar LF como separadores de líneas.

Durante las operaciones de importación y exportación, los caracteres de salto de línea se pueden convertir. Durante una importación, el analizador XML reemplaza los caracteres CRLF (rupturas de línea estándar en Windows) por caracteres de LF. Durante la exportación, los caracteres LF se sustituyen por caracteres CR en MacOs y por CRLF en Windows.

*Si desea mantener los retornos de carro, debe incluir en un elemento XML CDATA de manera que no sea procesado por el analizador XML. En lugar de caracteres CRLF, también puede utilizar el carácter "
", que es un retorno de carro explícito que no será procesado por el analizador.*

Ejemplo

Este ejemplo guarda el árbol vRefElem en el archivo MiDoc.xml:

```
DOM EXPORT TO FILE(vRefElem;"C:\\carpeta\MiDoc.xml")
```

Variables y conjuntos del sistema

Si el comando se ejecutó correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0 y se genera un error.

Gestión de errores

Se genera un error cuando:

- *La referencia del elemento no es válida,*
- *La ruta de acceso especificada no es válida,*
- *El volumen de almacenamiento devuelve un error (disco lleno, etc.).*

DOM EXPORT TO VAR

DOM EXPORT TO VAR (elementRef ; vXmlVar)

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML raíz
vXmlVar	Texto, BLOB	➡	Variable a recibir el árbol XML

Descripción

El comando **DOM EXPORT TO VAR** permite guardar un árbol XML en una variable texto o BLOB.

Pase en elementRef la referencia del elemento raíz a exportar.

Pase en vXmlVar el nombre de la variable que debe contener el árbol XML. Esta variable puede ser de tipo Texto o BLOB. Puede seleccionar el tipo en función de las operaciones a efectuar o del tamaño que el árbol pueda alcanzar (recuerde que en modo Unicode, las variables de tipo Texto están limitadas a 32 K de texto, mientras en modo Unicode, este límite es de 2 GB).

Recuerde que si utiliza una variable de tipo texto para almacenar el elemento refElement, en modo no Unicode, será codificado utilizando el conjunto de caracteres Mac "actual" (es decir, Mac Roman en la mayoría de los sistemas occidentales). Esto significa que el texto devuelto perderá su codificación original (encoding="xxx"). En este caso, la variables vVarXml permite visualizar o almacenar el código obtenido pero NO generar un documento XML válido (utilizando el comando **SEND PACKET** por ejemplo).

En modo Unicode, el código original se conserva en la variable.

Notas sobre el procesamiento de caracteres de fin de línea

En XML, los saltos de línea no son significativos, independientemente de si se encuentran dentro o entre los elementos XML. Internamente, XML utiliza caracteres estándar LF como separadores de líneas.

Durante las operaciones de importación y exportación, los caracteres de salto de línea se pueden convertir. Durante una importación, el analizador XML reemplaza los caracteres CRLF (rupturas de línea estándar en Windows) por caracteres de LF. Durante la exportación, los caracteres LF se sustituyen por caracteres CR en MacOs y por CRLF en Windows.

Si desea mantener los retornos de carro, debe incluir en un elemento XML CDATA de manera que no sea procesado por el analizador XML. En lugar de caracteres CRLF, también puede utilizar el carácter "
", que es un retorno de carro explícito que no será procesado por el analizador.

Ejemplo

Este ejemplo guarda el árbol vRefElem en una variable texto:

```
C_TEXT(vtMiTexto)
DOM EXPORT TO VAR(vRefElem;vtMiTexto)
```

Variables y conjuntos del sistema

Si el comando se ejecutó correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0 y se genera un error (por ejemplo, si la referencia del elemento no es válida).

DOM Find XML element

DOM Find XML element (elementRef ; xRuta {; arrRefElement}) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
xRuta	Texto	➡	Ruta XPath del elemento a buscar
arrRefElement	Array cadena	⬅	Lista de referencias de los elementos encontrados (si aplica)
Resultado	Cadena	↩	Referencia del elemento encontrado (si aplica)

Descripción

El comando ***DOM Find XML element*** *permite buscar los elementos XML específicos en una estructura XML. La búsqueda comienza por el elemento designado por el parámetro refElement.*

El nodo XML a buscar está definido por el parámetro xRuta, expresado en notación XPath (ver el párrafo "Uso de la notación XPath" en la sección ***Presentación de los comandos XML DOM****). Es posible utilizar elementos indexados.*

Nota: *conforme a la norma XML, la búsqueda diferencia las mayúsculas y las minúsculas.*

El comando devuelve la referencia XML del elemento encontrado.

Cuando se pasa el array cadena arrRefElement, el comando lo llena con la lista de las referencias XML encontradas. En este caso, el comando devuelve el primer elemento del array arrRefElement. Este parámetro es útil cuando existen varios elementos con el mismo nombre en la ubicación designada por el parámetro xRuta.

Ejemplo 1

Este ejemplo permite buscar rápidamente un elemento XML y mostrar su valor:

```
vEncontrado:=DOM Find XML element(vRefElem;"Items/Book[15]/Title")
DOM GET XML ELEMENT VALUE(vEncontrado;valor)
ALERT("El valor del elemento es: \""+valor+"\"")
```

La misma búsqueda también puede efectuarse de esta forma:

```
vEncontrado:=DOM Find XML element(vElemRef;"Items/Book[15]")
vEncontrado:=DOM Find XML element(vEncontrado;"Book/Title")
DOM GET XML ELEMENT VALUE(vEncontrado;valor)
ALERT("El valor del elemento es: \""+valor+"\"")
```

Nota: *como se muestra en el ejemplo anterior, la ruta XPath debe siempre comenzar por el nombre del elemento actual. Esta precisión es importante cuando maneja rutas XPath relativas.*

Ejemplo 2

Dada la siguiente estructura XML:

```
<Root>  <Elem1>  <Elem2>aaa</Elem2>  <Elem2>bbb</Elem2>  <Elem2>ccc</Elem2>  </Elem1> </Root>
```

El siguiente código permite recuperar la referencia de cada elemento Elem2 en el array arrEncontrados:

```
ARRAY TEXT(arrEncontrados;0)
vFound:=DOM Find XML element(vElemRef;"/Root/Elem1/Elem2";arrEncontrados)
```

Variables y conjuntos del sistema

Si el comando se ejecutó correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0 y se genera un error.

Gestión de errores

Se genera un error cuando:

- *La referencia del elemento no es válida*
- *La ruta xPath especificada no es válida.*

DOM Find XML element by ID

DOM Find XML element by ID (elementRef ; id) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	→	Referencia del elemento XML
id	Cadena	→	Valor del atributo ID del elemento a buscar
Resultado	Cadena	←	Referencia del elemento encontrado (si aplica)

Descripción

*El comando **DOM Find XML element by ID** busca, al interior de un documento XML, el elemento cuyo atributo id sea igual al valor pasado en el parámetro id.*

Pase en refElemento la referencia de un elemento del documento XML en el cual quiere realizar la búsqueda. Puede pasar la referencia del elemento raíz o de otro elemento, la búsqueda no tiene en cuenta la posición de refElemento y se efectúa siempre en la totalidad del documento.

El comando devuelve en resultado la referencia XML del elemento encontrado.

***Advertencia**: en XML, el atributo id asocia un identificador único a cada elemento del documento. El valor del atributo id debe ser un nombre XML válido y debe ser único en el documento XML (restricción de validez). Para que el comando **DOM Find XML element by ID** funcione correctamente, esta restricción debe respetarse; de lo contrario el resultado es aleatorio (el comando devuelve la referencia al primer elemento encontrado en el documento).*

DOM Get first child XML element

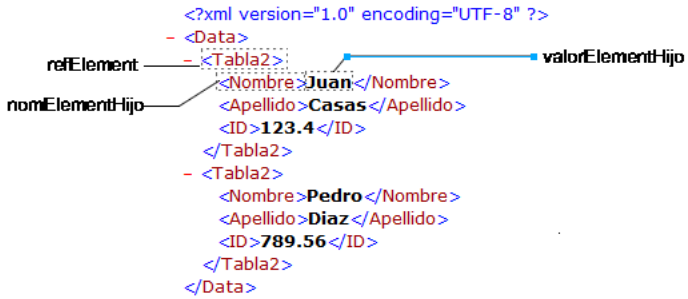
DOM Get first child XML element (elementRef {; nomElementHijo {; valorElementHijo}}) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
nomElementHijo	Cadena	⬅	Nombre del elemento XML hijo
valorElementHijo	Cadena	⬅	Valor del elemento XML hijo
Resultado	Cadena	➲	Referencia del elemento XML hijo

Descripción

*El comando **DOM Get first child XML element** devuelve una referencia XML al primer "hijo" del elemento XML pasado en refElement. Esta referencia puede utilizarse con otros comandos de análisis XML.*

Los parámetros nomElementHijo y valorElemHijo, si se pasan, reciben respectivamente el nombre y el valor del elemento hijo.



Ejemplo 1

Recuperación de la referencia del primer elemento XML del padre raíz. La estructura XML (C:\\import.xml) se carga previamente en un BLOB:

```
C_BLOB(miVarBlob)
C_TEXT($xml_Padre_Ref;$xml_Hijo_Ref)

DOCUMENT TO BLOB("c:\\importar.xml";miVarBlob)
$xml_Padre_Ref:=DOM Parse XML variable(miVarBlob)
$xml_Hijo_Ref:=DOM Get first child XML element($xml_Padre_Ref)
```

Ejemplo 2

Recuperación de la referencia, del nombre y del valor del primer elemento XML del padre raíz. La estructura XML (C:\\importar.xml) se carga previamente en un BLOB:

```
C_BLOB(miVarBlob)
C_TEXT($xml_Padre_Ref;$xml_Hijo_Ref)
C_TEXT($hijoNom;$hijoValor)

DOCUMENT TO BLOB("c:\\importar.xml";miVarBlob)
$xml_Padre_Ref:=DOM Parse XML variable(miVarBlob)
$xml_Hijo_Ref:=DOM Get first child XML element($xml_Padre_Ref;$hijoNom;$hijoValor)
```

Variables y conjuntos del sistema

Si el comando se ejecutó correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0.

DOM Get last child XML element

DOM Get last child XML element (elementRef {; nomElementHijo {; valorElementHijo}}) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
nomElementHijo	Cadena	⬅	Nombre del elemento hijo
valorElementHijo	Cadena	⬅	Valor del elemento hijo
Resultado	Cadena	➡	Referencia del elemento XML

Descripción

*El comando **DOM Get last child XML element** devuelve una referencia XML al último "hijo" del elemento XML pasado como referencia en refElement. Esta referencia puede utilizarse con otros comandos de análisis XML.*

Los parámetros opcionales nomElementHijo y valorElementHijo, si se pasan, reciben respectivamente el nombre y el valor del elemento "hijo".

Ejemplo

Recuperación de la referencia del último elemento XML del padre raíz. La estructura XML (C:\\importar.xml) se carga previamente en un BLOB:

```
C_BLOB(miVarBlob)
C_TEXT($ref_XML_Padre;$ref_XML_Hijo)
C_TEXT($nombreHijo;$valorHijo)

DOCUMENT TO BLOB("c:\\importar.xml";miVarBlob)
$ref_XML_Padre:=DOM Parse XML variable(miVarBlob)
$ref_XML_Hijo:=DOM Get last child XML element($ref_XML_Padre;$nombreHijo;$valorHijo)
```

Variables y conjuntos del sistema

Si el comando se ejecutó correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0.

DOM Get next sibling XML element

DOM Get next sibling XML element (elementRef {; nomElemHermano {; valorElemHermano}}) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
nomElemHermano	Cadena	⬅	Nombre del elemento XML hermano
valorElemHermano	Cadena	⬅	Valor del elemento XML hermano
Resultado	Cadena	➲	Referencia del elemento XML hermano

Descripción

*El comando **DOM Get next sibling XML element** devuelve una referencia al próximo "hermano" del elemento XML pasado como referencia. Esta referencia puede utilizarse con otros comandos de análisis XML.*

Los parámetros nomElemHermano y valorElemHermano, si se pasan, reciben respectivamente el nombre y el valor del elemento "hermano".

Este comando se utiliza para navegar entre los "hijos" de un elemento XML.

Después del último "hermano," la variable sistema OK toma el valor 0.

Ejemplo 1

Recuperación de la referencia del elemento XML hermano seguido por el elemento pasado como parámetro:

```
C_TEXT($xml_Padre_Ref;$siguiente_XML_Ref)
$siguiente_XML_Ref:=DOM Get next sibling XML element($xml_Padre_Ref)
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Data>
  - <Tabla2>                          Elemento referenciado
      <Nombre>Juan</Nombre>
      <Apellido>Casas</Apellido>
      <ID>123.4</ID>
    </Tabla2>
  - <Tabla2>                          Elemento hermano siguiente
      <Nombre>Pedro</Nombre>
      <Apellido>Diaz</Apellido>
      <ID>789.56</ID>
    </Tabla2>
</Data>
```

Ejemplo 2

Recuperación en un bucle de las referencias de todos los elementos XML hijos del elemento padre pasado como parámetro, comenzando con el primer hijo:

```
C_TEXT($xml_Padre_Ref;$primer_XML_Ref;$siguiente_XML_Ref)

$primer_XML_Ref:=DOM Get first child XML element($xml_Padre_Ref)
$siguiente_XML_Ref:=$primer_XML_Ref
While(OK=1)
   $siguiente_XML_Ref:=DOM Get next sibling XML element($siguiente_XML_Ref)
End while
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Data>
  - <Tabla2>                          Elemento referenciado
      <Nombre>Juan</Nombre>           ——— Primer elemento hijo
      <Apellido>Casas</Apellido>      ——— Elemento hermano siguiente (bucle1)
      <ID>123.4</ID>                  ——— Elemento hermano siguiente (bucle 2)
    </Tabla2>
  - <Tabla2>
      <Nombre>Pedro</Nombre>
      <Apellido>Diaz</Apellido>
      <ID>789.56</ID>
    </Tabla2>
</Data>
```

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente y si el elemento analizado no es el último "hermano" del elemento referenciado, la variable sistema OK toma el valor 1. Si ocurre un error o si el elemento analizado es el último "hermano" del elemento referenciado, toma el valor 0.

DOM Get parent XML element

DOM Get parent XML element (elementRef {; nomElemPadre {; valorElemPadre}}) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
nomElemPadre	Cadena	⬅	Nombre del elemento XML padre
valorElemPadre	Cadena	⬅	Valor del elemento XML padre
Resultado	Cadena	➲	Referencia del elemento XML padre

Descripción

*El comando **DOM Get parent XML element** devuelve una referencia XML al "padre" del elemento XML pasado como referencia en refElement. Esta referencia puede ser utilizada con los otros comandos de análisis XML.*

Los parámetros opcionales nomElemPadre y valorElemPadre, cuando se pasan, reciben respectivamente el nombre y el valor del elemento padre.

Si utiliza este comando en un nodo documento

Cuando pasa un elemento raíz en refElement, el comando devuelve la referencia "#document". El nodo documento es el padre de un elemento raíz.
Si utiliza este comando en un nodo documento, el comando devuelve una referencia nula ("0000000000000000") y la variable OK toma el valor 0.

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0.

DOM Get previous sibling XML element

DOM Get previous sibling XML element (elementRef {; nomElemHermano {; valorElemHermano}}) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	→	Referencia del elemento XML
nomElemHermano	Cadena	←	Nombre del elemento XML hermano
valorElemHermano	Cadena	←	Valor del elemento XML hermano
Resultado	Cadena	↩	Referencia del elemento XML hermano

Descripción

*El comando **DOM Get previous sibling XML element** devuelve una referencia al anterior "hermano" del elemento XML pasado en referencia. Esta referencia puede utilizarse con los otros comandos de análisis XML.*

Los parámetros opcionales nomElemHermano y valorElemHermano, si se pasan, reciben respectivamente el nombre y el valor del elemento "hermano" anterior.

Este comando puede utilizarse para navegar entre los "hijos" de un elemento XML.

Antes del primer "hermano," la variable sistema OK tiene el valor 0.

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente y si el elemento referenciado no es el primer "hijo" de la estructura, la variable sistema OK toma el valor 1. Si ocurre un error o si el elemento analizado es el primer "hijo" de la estructura, toma el valor 0.

DOM Get Root XML element

DOM Get Root XML element (elementRef) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
Resultado	Cadena	⮌	Referencia del elemento raíz o "" en caso de error

Descripción

*El comando **DOM Get Root XML element** devuelve una referencia al elemento raíz del documento al cual pertenece el elemento XML que se pasa en el parámetro elementRef. Esta referencia puede utilizarse con los otros comandos de análisis XML.*

DOM GET XML ATTRIBUTE BY INDEX

DOM GET XML ATTRIBUTE BY INDEX (elementRef ; indexAtrib ; nomAtrib ; valorAtrib)

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
indexAtrib	Entero largo	➡	Número de índice del atributo
nomAtrib	Variable	⬅	Nombre del atributo
valorAtrib	Variable	⬅	Valor del atributo

Descripción

*El comando **DOM GET XML ATTRIBUTE BY INDEX** permite conocer el nombre así como el valor de un atributo especificado por su número de índice.*

Pase en refElement la referencia de un elemento XML y en indexAtrib el número de índice del atributo que quiere conocer el nombre. El nombre se devuelve en el parámetro nomAtrib y su valor se devuelve en el parámetro valorAtrib. 4D intentará convertir el valor obtenido en el tipo de variable pasada como parámetro.

***Nota:** el número de índice no corresponde a la ubicación del atributo en el archivo XML mostrado en forma de texto. En XML, el índice de un atributo indica su posición entre los atributos clasificados por orden alfabético (en función de su nombre). Para ver una ilustración de este principio, consulte el ejemplo del comando **DOM Count XML attributes**.*

Si el valor pasado en indexAtrib es superior al número de atributos presentes en el elemento XML, se devuelve un error.

Ejemplo

*Consulte el ejemplo del comando **DOM Count XML attributes**.*

Variables y conjuntos del sistema

Si el comando ha sido ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM GET XML ATTRIBUTE BY NAME

DOM GET XML ATTRIBUTE BY NAME (elementRef ; nomAtrib ; valorAtrib)

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
nomAtrib	Cadena	➡	Nombre del atributo
valorAtrib	Variable	⬅	Valor del atributo

Descripción

*El comando **DOM GET XML ATTRIBUTE BY NAME** permite conocer el valor de un atributo especificado por su nombre.*

Pase en refElement la referencia de un elemento XML y en nomAtrib el nombre del atributo del que quiere conocer su valor. El valor se devuelve en el parámetro valorAtrib.

4D intentará convertir el valor obtenido en el tipo de la variable pasada como parámetro.

Si no existe ningún atributo nomAtrib en el elemento XML, se devuelve un error. Si varios atributos del elemento XML tienen el mismo nombre, sólo se devuelve el valor del primer atributo.

Ejemplo

Este método se utiliza para recuperar un valor de atributo XML utilizando su nombre:

```
C_BLOB(miVarBlob)
C_TEXT($ref_XML_Padre;$ref_XML_Hijo)
C_LONGINT($NumLinea)

$ref_XML_Padre:=DOM Parse XML variable(miVarBlob)
$ref_XML_Hijo:=DOM Get first child XML element($ref_XML_Padre)
DOM GET XML ATTRIBUTE BY NAME($ref_XML_Hijo;"N";$NumLinea)
```

Si este método se aplica al ejemplo a continuación, $NumLinea contiene el valor 1:

```
<?xml version="1.0" ?>
- <STANZA>
    <LINE N="1">I heard a thousand blended notes,</LINE>
    <LINE N="2">While in grove I sate reclined,</LINE>
    <LINE N="3">In that sweet mood when pleasant thoughts</LINE>
    <LINE N="4">Bring sad thoughts to the mind.</LINE>
  </STANZA>
```

Variables y conjuntos del sistema

Si el comando ha sido ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM GET XML CHILD NODES

DOM GET XML CHILD NODES (refElement ; ArrTiposHijos ; arrRefsNodos)

Parámetro	Tipo		Descripción
refElement	Texto	➡	Referencia del elemento XML
ArrTiposHijos	Array entero largo	⬅	Tipos de nodos hijos
arrRefsNodos	Array texto	⬅	Referencias o Valores de los nodos hijos

Descripción

*El comando **DOM GET XML CHILD NODES** devuelve los tipos y referencias o valores de todos los nodos hijos del elemento XML designado por refElement.*

Los tipos de los nodos hijos se devuelven en el array arrTiposHijos. Puede comparar los valores devueltos por el comando con las siguientes constantes del tema "":

Constante	Tipo	Valor
XML comment	Entero largo	2
XML processing instruction	Entero largo	3
XML DATA	Entero largo	6
XML CDATA	Entero largo	7
XML DOCTYPE	Entero largo	10
XML ELEMENT	Entero largo	11

*Para mayor información, consulte la descripción del comando **DOM Append XML child node**.*

El array arrRefsNodos recibe los valores o las referencias de los elementos en función de su naturaleza (contenidos o instrucciones).

Ejemplo

Dada la siguiente estructura XML:

```
<myElement>Hola<br/>Nueva<br/>York</myElement>
```

Después de la ejecución de estas instrucciones:

```
elementRef:=DOM Find XML element($root;"myElement")
DOM GET XML CHILD NODES(elementRef;$typeArr;$textArr)
```

... los arrays $typeArr y $textArr contendrán los siguientes valores:

$typeArr{1}=6 $textArr{1} = "Hola"
*$typeArr{2}=11 $textArr{2} = "AEF1233456878977" (element reference
)*
$typeArr{3}=6 $textArr{3} = "Nueva"
*$typeArr{4}=11 $textArr{4} = "AEF1237897734568" (element reference
)*
$typeArr{5}=6 $textArr{5} = "York"

DOM Get XML document ref

DOM Get XML document ref (refElement) -> Resultado

Parámetro	Tipo		Descripción
refElement	Texto	→	Referencia de un elemento existente en un árbol DOM
Resultado	Texto	←	Referencia del primer elemento del árbol DOM (nodo document)

Descripción

*El comando **DOM Get XML document ref** se utiliza para recuperar la referencia del elemento "documento" del árbol DOM cuya referencia se pasa en refElement. El elemento document es el primer elemento de un árbol DOM; es el padre del elemento raíz.*

*La referencia del elemento document permite manipular los nodos "Doctype" y "Processing Instruction". Sólo puede utilizarse con los comandos **DOM Append XML child node** y **DOM GET XML CHILD NODES**.*

A este nivel, puede únicamente añadir las instrucciones y comentarios o reemplazar el nodo Doctype. No puede crear nodos CDATA o Text.

Ejemplo

En este ejemplo, queremos encontrar la declaración DTD del documento XML:

```
C_TEXT($rootRef)
$rootRef:=DOM Parse XML source("")
If(OK=1)
   C_TEXT($documentRef)
 // estamos buscando el nodo document, ya que es el nodo al cual está
 // asociado el nodo DOCTYPE antes del nodo raíz
   $documentRef:=DOM Get XML document ref($rootRef)
   ARRAY TEXT($typeArr;0)
   ARRAY TEXT($valueArr;0)
 // en este nodo buscamos entre los hijos el nodo de tipo DOCTYPE
   DOM GET XML CHILD NODES($refDocument;$typeArr;$valueArr)
   C_TEXT($text)
   $text:=""
   $pos:=Find in array($typeArr;XML DOCTYPE)
   If($pos>-1)
 // Recuperamos en $text la declaración de DTD
      $text:=$text+"Doctype: "+$valueArr{$pos}+Char(Carriage return)
   End if
   DOM CLOSE XML($rootRef)
End if
```

DOM Get XML element

DOM Get XML element (elementRef ; nomElement ; indice ; valorElement) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
nomElement	Cadena	➡	Nombre del elemento a leer
indice	Entero largo	➡	Número de índice del elemento a leer
valorElement	Variable	⬅	Valor del elemento
Resultado	Cadena	↩	Referencia del elemento XML (16 caracteres)

Descripción

El comando ***DOM Get XML element*** *devuelve una referencia XML al elemento "hijo" dependiente de los parámetros nomElement e index.*

El valor del elemento también se devuelve en el parámetro valorElement.

Nota: *por defecto,* ***DOM Get XML element*** *es sensible a las mayúsculas y minúsculas en relación con el parámetro nomElement (de acuerdo a los estándares xml). Puede controlar la sensibilidad de mayúsculas y minúsculas del comando utilizando el selector XML DOM case sensitivity del comando* ***XML SET OPTIONS****.*

Variables y conjuntos del sistema

Si el comando ha sido ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM GET XML ELEMENT NAME

DOM GET XML ELEMENT NAME (elementRef ; nomElement)

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
nomElement	Variable	⬅	Nombre del elemento

Descripción

*El comando **DOM GET XML ELEMENT NAME** devuelve en el parámetro nomElement, el nombre del elemento XML designado por refElement. Para mayor información sobre los nombres de elementos XML, consulte la sección **Presentación de los comandos XML DOM**.*

Ejemplo

Este método devuelve el nombre del elemento $xml_Element_Ref:

```
C_TEXT($xml_Element_Ref)
C_TEXT($nom)

DOM GET XML ELEMENT NAME($xml_Element_Ref;$nom)
```

Variables y conjuntos del sistema

Si el comando ha sido ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM GET XML ELEMENT VALUE

DOM GET XML ELEMENT VALUE (elementRef ; valorElement {; cDATA})

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
valorElement	Variable	⬅	Valor del elemento
cDATA	Variable	⬅	Contenido de la sección CDATA

Descripción

*El comando **DOM GET XML ELEMENT VALUE** devuelve en el parámetro valorElement, el valor del elemento XML designado por refElement. 4D intentará convertir el valor obtenido en el tipo de variable pasada como parámetro.*

El parámetro opcional cDATA se utiliza para recuperar los contenidos de las secciones CDATA del elemento XML refElement. Como con el parámetro valorElement, 4D intentará convertir el valor obtenido en el tipo de la variable pasada como parámetro.

***Nota:** si el elemento designado por refElement es un BLOB procesado por el comando **DOM SET XML ELEMENT VALUE**, ha sido codificado automáticamente en base64. Por lo tanto, el comando intentará automáticamente de decodificar en base 64.*

Ejemplo

Este método devuelve el valor del elemento $xml_Element_Ref:

```
C_TEXT($xml_Element_Ref)
C_REAL($valor)

DOM GET XML ELEMENT VALUE($xml_Element_Ref;$valor)
```

Variables y conjuntos del sistema

Si el comando se ha ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM Get XML information

DOM Get XML information (elementRef ; infoXML) -> Resultado

Parámetro	Tipo		Descripción
elementRef	Cadena	→	Referencia del elemento XML raíz
infoXML	Entero largo	→	Tipo de información a obtener
Resultado	Cadena	↩	Valor de la información XML

Descripción

*El comando **DOM Get XML information** permite recuperar diversa información sobre el elemento XML designado por refElement.*

En infoXML, pase un código indicando el tipo de información a recuperar. Puede utilizar las siguientes constantes predefinidas, ubicadas en el tema "":

Constante	Tipo	Valor	Comentario
PUBLIC ID	Entero largo	1	Identificador público (FPI) de la DTD a la cual el documento es conforme (si la etiqueta DOCTYPE xxx PUBLIC está presente).
SYSTEM ID	Entero largo	2	Identificador sistema
DOCTYPE Name	Entero largo	3	Nombre del elemento raíz tal como se definió en el marcador DOCTYPE
Encoding	Entero largo	4	Codificación utilizada (UTF-8, ISO...)
Version	Entero largo	5	Versión de XML aceptada
Document URI	Entero largo	6	URI de la DTD

Estas constantes indican la siguiente información:

- *PUBLIC ID: identificador público (FPI) de la DTD a la cual el documento se conforma (si la etiqueta DOCTYPE xxx PUBLIC está presente).*
- *SYSTEM ID: identificador sistema.*
- *Nombre DOCTYPE: nombre del elemento raíz tal como se definió en la etiqueta DOCTYPE.*
- *Encoding: codificación utilizada (UTF-8, ISO...).*
- *Version: versión de XML aceptada.*
- *Document URI: URI de la DTD.*

DOM Insert XML element

DOM Insert XML element (refElementTarget ; refElementFuente ; indiceHijo) -> Resultado

Parámetro	Tipo		Descripción
refElementTarget	Texto	➡	Referencia del elemento XML padre
refElementFuente	Texto	➡	Referencia del elemento XML a insertar
indiceHijo	Entero largo	➡	Índice del hijo del elemento objetivo antes del cual se debe insertar el nuevo elemento
Resultado	Texto	⮌	Referencia del nuevo elemento XML

Descripción

*El comando **DOM Insert XML element** se utiliza para insertar un nuevo elemento XML entre los hijos del elemento XML cuya referencia se pasa en el parámetro refElementTarget.*

Pase el elemento a insertar en refElementFuente. Este elemento se debe pasar como la referencia de un elemento XML existente en un árbol DOM.

El parámetro indiceHijo se puede utilizar para designar al hijo del elemento padre antes del cual se debe insertar el nuevo elemento. Pase un número de índice en este parámetro. Si el valor no es válido (por ejemplo, no hay ningún elemento hijo de este índice), el nuevo elemento se agrega antes del primer hijo del elemento padre.

El comando devuelve la referencia del elemento XML obtenido.

Ejemplo

En la siguiente estructura, queremos invertir el primer y segundo libro:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<BookCatalog>  <Book>  <Title>Open Source Web Services</Title>  <Author>Collective</Author>
<Date>2003</Date>  <ISBN>2-7440-1507-5</ISBN>  <Publisher>Wrox</Publisher>  </Book> <Book>
<Title>Building XML Web services</Title>  <Author>Scott Short</Author>  <Date>2002</Date>  <ISBN>2-10-
006476-2</ISBN>  <Publisher>Microsoft Press</Publisher>  </Book> </BookCatalog>
```

Para hacerlo, simplemente ejecutamos el siguiente código:

```
C_TEXT($rootRef)
<p>$rootRef:=DOM Parse XML source("") //selección de documento XML
If(OK=1)
   C_TEXT($newStruct)
   $newStruct:=DOM Create XML Ref("BookCatalog")

   $bookRef:=DOM Find XML element($rootRef;"/BookCatalog/Book[1]")
   $newElementRef:=DOM Append XML element($newStruct;$bookRef)

   $bookRef:=DOM Find XML element($rootRef;"/BookCatalog/Book[2]")
   C_TEXT($newElementRef)
   $newElementRef:=DOM Insert XML element($newStruct;$bookRef;1)

   DOM CLOSE XML($newStruct)
   DOM CLOSE XML($rootRef)
End if
```

DOM Parse XML source

DOM Parse XML source (doc {; validacion {; dtd | esquema}}) -> Resultado

Parámetro	Tipo		Descripción
doc	Cadena	➡	Ruta de acceso del documento
validacion	Booleano	➡	True = Validación False = No validación
dtd | esquema	Cadena	➡	Ubicación de la DTD o del esquema XML
Resultado	Cadena	⬅	Referencia del elemento XML

Descripción

*El comando **DOM Parse XML source** analiza un documento que contiene una estructura XML y devuelve una referencia para este documento. El comando puede validar o no el documento vía una DTD o un esquema XML (documento XSD, XML Schema Definition).*

El documento puede estar en el disco o en Internet/Intranet.

***Nota:** la ejecución del comando **DOM Parse XML source** es síncrona.*

En el parámetro documento puede pasar:

- *una ruta de acceso completa estándar (del tipo C:\\Carpeta\\Archivo\\... bajo Windows y MacintoshHD:Carpeta:Archivo bajo Mac OS),*
- *o una ruta Unix bajo Mac OS (la cual debe comenzar por /).*
- *o una ruta red de tipo http://www.sitio.com/Archivo o ftp://publico.ftp.com...*

El parámetro booleano validacion permite indicar si validar o no la estructura.

- *Si validacion es igual a True, la estructura se validará. En este caso, el analizador intentará validar la estructura XML del documento basado en la referencia DTD o el esquema XML incluida en el documento, o vía la DTD o el esquema XML designado por el tercer parámetro, si se pasa.*
- *Si validacion es igual a False, la estructura no se validará.*

Si pasa True en validacion y omite el tercer parámetro, el comando intentará validar la estructura XML vía una referencia DTD o XSD que se encuentra en la estructura misma. La validación puede ser indirecta: si la estructura contiene una referencia a un archivo DTD que contiene una referencia a un archivo XSD, el comando intentará efectuar las dos validaciones.
El tercer parámetro permite designar una DTD específica o un esquema XML para el análisis del documento. Si utiliza este parámetro, el comando no tendrá en cuenta la DTD referenciada en el documento XML.

Validación por DTD

Hay dos formas de especificar una DTD:

- *Como una referencia. Para hacerlo, pase la ruta de acceso completa de la nueva DTD (extensión "dtd") en el parámetro dtd. Si el documento indicado no contiene una DTD válida, el parámetro dtd se ignora y se genera un error.*
- *Directamente en un texto. En este caso, si los contenidos del parámetro comienzan por "<?xml", 4D considerará que esa es la DTD; de lo contrario, 4D lo considerará como una ruta de acceso.*

Validación por esquema

Para validar el documento vía un esquema XML, sólo necesita pasar en el tercer parámetro un archivo o un URL de extensión "xsd" en lugar de un "dtd". La validación por esquema XML se considera más flexible y poderosa que la validación por DTD. El lenguaje de los documentos XSD está basado en el lenguaje XML. Más particularmente, los esquemas XML soportan tipos de datos. Para mayor información sobre los esquemas XML, por favor consulte la siguiente dirección: http://www.w3.org/XML/Schema.

*Si la validación no se puede efectuar (no DTD o XSD, URL incorrecto, etc.), se genera un error. La variable sistema Error indica el número del error. Puede interceptar este error utilizando un método instalado por el comando **ON ERR CALL**.*

El comando devuelve una cadena de 16 caracteres (RefElement) constituyendo la referencia en memoria de la estructura virtual del documento. Esta referencia debe utilizarse con los otros comandos de análisis XML.

***Importante:** una vez no lo necesite, recuerde llamar al comando **DOM CLOSE XML** con esta referencia con el fin de liberar memoria.*

Ejemplo 1

Apertura sin validación de un documento XML en disco:

```
$xml_Ref_Estruct:=DOM Parse XML source("C:\\importar.xml")
```

Ejemplo 2

Apertura sin validación de un documento XML ubicado junto al archivo de estructura de la base:

```
$xml_Ref_Estruct:=DOM Parse XML source("importar.xml")
```

Ejemplo 3

Apertura de un documento XML ubicado en el disco y validación utilizando un DTD ubicado en el disco:

```
$xml_Ref_Estruct:=DOM Parse XML source("C:\\importar.xml";True;"C:\\importar_dtd.xml")
```

Ejemplo 4

Apertura sin validación de un documento XML ubicado en un URL específico:

```
$xml_Ref_Estruct:=DOM Parse XML source("http://www.4DHispano.com/xml/importar.xml")
```

Variables y conjuntos del sistema

Si el comando ha sido ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM Parse XML variable

DOM Parse XML variable (variable {; validacion {; dtd | esquema}}) -> Resultado

Parámetro	Tipo		Descripción
variable	BLOB, Texto	➡	Nombre de la variable
validacion	Booleano	➡	True = Validación por la DTD, False = No validacion
dtd | esquema	Cadena	➡	Ubicación de la DTD o del esquema XML
Resultado	Cadena	⮌	Referencia del elemento XML

Descripción

*El comando **DOM Parse XML variable** analiza una variable de tipo BLOB o Texto que contiene una estructura XML y devuelve una referencia para esta variable. El comando puede validar o no la estructura vía un DTD o un esquema XML (XML Schema Definition (XSD) document).*

Pase en el parámetro variable el nombre de la variable BLOB o el Texto que contiene el objeto XML.

El parámetro booleano validacion permite indicar si validar o no la estructura utilizando la DTD.

- *Si validacion es igual a True, la estructura se validará. En este caso, el analizador intentará validar la estructura XML del documento basado en la referencia DTD o XSD incluida en el documento, o vía la DTD o el esquema XML designado por el tercer parámetro si se pasa.*
- *Si validacion es igual a False, la estructura no se validará.*

Si pasa True en validacion y omite el tercer parámetro, el comando intentará validar la estructura XML vía una referencia DTD o XSD que se encuentra en la estructura misma. La validación puede ser indirecta: si la estructura contiene una referencia a un archivo DTD que contiene a su vez una referencia a un archivo XSD, el comando intentará efectuar ambas validaciones.

El tercer parámetro dtd, permite indicar una DTD específica o un esquema XML para el análisis del documento. Si utiliza este parámetro, el comando no tendrá en cuenta la DTD referenciada en el documento XML.

Validación por DTD
Hay dos formas de especificar un DTD:

- *como una referencia. Para hacer esto, pase la ruta de acceso completa de la nueva DTD (extensión "dtd) en el parámetro dtd. Si el documento indicado no contiene una DTD válida, el parámetro dtd se ignora y se genera un error.*
- *directamente en un texto. En este caso, si el contenido del parámetro comienza por "<?xml", 4D lo considerará como la DTD; de lo contrario, 4D lo considerará como una ruta de acceso.*

Validación por esquema
Para validar el documento vía un esquema XML, sólo debe pasar un archivo o URL con una extensión "xsd" en lugar de una "dtd" en el tercer parámetro. La validación por esquema XML se considera más flexible y poderosa que la validación por DTD. El lenguaje de documentos XSD está basado en lenguaje XML. Los esquemas XML soportan particularmente tipos de datos. Para mayor información sobre los esquemas XML, consulte la siguiente dirección: http://www.w3.org/XML/Schema.

*Si no se puede efectuar la validación (no DTD o XSD, URL incorrecto, etc.), se genera un error. La variable sistema Error indica el número del error. Puede interceptar este error con la ayuda de un método instalado por el comando **ON ERR CALL**.*

El comando devuelve una cadena de caracteres (RefElement) que constituye la referencia en memoria de la estructura virtual de la variable. Esta referencia debe utilizarse con otros comandos de análisis XML.

***Importante:** una vez no lo necesite más, recuerde llamar al comando **DOM CLOSE XML** con esta referencia para liberar memoria.*

Ejemplo 1

Apertura sin validación de un objeto XML ubicado en una variable Texto 4D:

```
C_TEXT(myTextVar)
C_TIME(vDoc)
C_TEXT($xml_Struct_Ref)

vDoc:=Open document("Document.xml")
If(OK=1)
   RECEIVE PACKET(vDoc;myTextVar;32000)
   CLOSE DOCUMENT(vDoc)
   $xml_Struct_Ref:=DOM Parse XML variable(myTextVar)
End if
```

Ejemplo 2

Apertura sin validación de un documento XML ubicado en un BLOB 4D:

```
C_BLOB(myBlobVar)
C_TEXT($ref_XML_Struct)
```

```
DOCUMENT TO BLOB(c:\import.xml;myBlobVar)
$xml_Struct_Ref:=DOM Parse XML variable(myBlobVar)
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0.

DOM REMOVE XML ATTRIBUTE

DOM REMOVE XML ATTRIBUTE (refElement ; nomAtrib)

Parámetro	Tipo		Descripción
refElement	Texto	➡	Referencia del elemento XML
nomAtrib	Texto	➡	Atributo a borrar

Descripción

*El comando **DOM REMOVE XML ATTRIBUTE** remueve, si existe, el atributo designado por nomAtrib del elemento XML cuya referencia se pasa en el parámetro refElement.*

Si atributo se suprime correctamente, la variable sistema OK toma el valor 1. Si no existe ningún atributo llamado nomAtrib en refElement, se devuelve un error y la variable sistema OK toma el valor 0.

Ejemplo

Dada la siguiente estructura:

```
<?xml version="1.0" ?>
- <STANZA>
    <LINE N="1">I heard a thousand blended notes,</LINE>
    <LINE N="2">While in grove I sate reclined,</LINE>
    <LINE N="3">In that sweet mood when pleasant thoughts</LINE>
    <LINE N="4">Bring sad thoughts to the mind.</LINE>
  </STANZA>
```

El siguiente código permite remover el primer atributo "N=1":

```
C_BLOB(myBlobVar)
C_TEXT($xml_Parent_Ref;$xml_Child_Ref)
C_LONGINT($LineNum)

$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get first child XML element($xml_Parent_Ref)
DOM REMOVE XML ATTRIBUTE($xml_Child_Ref;"N")
```

DOM REMOVE XML ELEMENT

DOM REMOVE XML ELEMENT (elementRef)

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML

Descripción

El comando ***DOM REMOVE XML ELEMENT*** *elimina el elemento designado por* elementRef*.*

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario, toma el valor 0 y se genera un error.

Un error se genera cuando la referencia del elemento no es válida.

DOM SET XML ATTRIBUTE

DOM SET XML ATTRIBUTE (elementRef ; nomAtrib ; valorAtrib {; nomAtrib2 ; valorAtrib2 ; ... ; nomAtribN ; valorAtribN})

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
nomAtrib	Cadena	➡	Atributo a definir
valorAtrib	Cadena, Booleano, Entero largo, Real, Hora, Fecha	➡	Nuevo valor del atributo

Descripción

El comando **DOM SET XML ATTRIBUTE** permite añadir uno o varios atributos al elemento XML cuya referencia se pasa en el parámetro refElement. También permite definir el valor de cada atributo definido.

Pase en los parámetros nomAtrib y valorAtrib respectivamente el atributo a escribir y su valor (en forma de variables, campos, o valores literales). Puede pasar tantos atributos/valores como quiera.

El parámetro valorAtrib puede ser de tipo texto o de otro tipo (Booleano, entero, real, fecha u hora). Si pasa un valor de un tipo diferente a texto, 4D se encarga de su conversión a texto, de acuerdo a los siguientes principios:

Tipo	Ejemplo de valor convertido
Booleano	"true" o "false" (no traducido)
Entero	"123456"
Real	"12.34" (el separador decimal siempre es ".")
Fecha	"2006-12-04T00:00:00Z" (estándar RFC 3339)
Hora	"5233" (número de segundos)

Ejemplo

En la siguiente fuente XML:

```
<Book>  <Title>El mejor vendedor</Title> </Book>
```

Si se ejecuta el código siguiente:

```
vAtrNom:="Font"
vAtrVal:="Verdana"
DOM SET XML ATTRIBUTE(vRefElem;vAtrNom;vAtrVal)
```

Obtenemos:

```
<Book>  <Title Font=Verdana>El mejor vendedor</Title> </Book>
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. Si no, toma el valor 0 y se genera un error.

DOM SET XML DECLARATION

DOM SET XML DECLARATION (elementRef ; codificacion {; autonomo {; indentacion}})

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
codificacion	Cadena	➡	Conjunto de caracteres del documento XML
autonomo	Booleano	➡	True = el documento es autónomo False (por defecto) = el documento no es autónomo
indentacion	Booleano	➡	** Obsoleto no utilizar **

Descripción

El comando ***DOM SET XML DECLARATION*** *permite definir diferentes opciones que útiles en la creación del árbol XML designado por elementRef. Estas opciones hacen referencia a la codificación y a la propiedad autónoma del árbol:*

- *codificacion: indica el conjunto de caracteres utilizado en el documento. Por defecto (si no se llama al comando), se utiliza el conjunto de caracteres UTF-8 (Unicode comprimido).*
  Nota: *si pasa un conjunto de caracteres que no soporta los comandos XML de 4D, se utilizará UTF-8. Consulte* ***Conjuntos de caracteres*** *para ver la lista de conjuntos de caracteres soportados (sin embargo se recomienda UTF-8 en la mayoría de los casos).*
- *autonomo: indica si el árbol es autónomo (****True****) o si necesita otros archivos o recursos externos para su funcionamiento (****False****). Por defecto (si el comando no se llama o si se omite el parámetro), el árbol no es autónomo.*

Nota de compatibilidad: *El parámetro indentacion se conserva por razones de compatibilidad con las versiones anteriores de 4D pero su uso no se recomienda en 4D v12. De ahora en adelante, para especificar la indentación del documento, se recomienda utilizar el comando* ***XML SET OPTIONS****.*

Ejemplo

El siguiente ejemplo define la codificación a utilizar y la opción autónomo del elemento elementRef:

```
DOM SET XML DECLARATION(elementRef;"UTF-16";True)
```

DOM SET XML ELEMENT NAME

DOM SET XML ELEMENT NAME (elementRef ; nomElement)

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
nomElement	Cadena	➡	Nuevo nombre del elemento

Descripción

El comando **DOM SET XML ELEMENT NAME** permite modificar el nombre del elemento designado por refElement.
Pase en refElement la referencia del elemento a renombrar y en nomElement el nuevo nombre del elemento. El comando también se encarga de actualizar las etiquetas de apertura y cierre del elemento.

Ejemplo

En el siguiente recurso XML:

```
<Book>  <Title>El mejor vendedor</Title> </Book>
```

Si se ejecuta el siguiente código, suponiendo que vRefElem contiene la referencia del elemento 'Book':

```
DOM SET XML ELEMENT NAME(vRefElem;"MejorVendedor")
```

Obtenemos:

```
<MejorVendedor>  <Title>El mejor vendedor</Title> </MejorVendedor>
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

Gestión de errores

Se genera un error cuando:

- La referencia del elemento no es válida
- El nuevo nombre del elemento a crear no es válido (por ejemplo, si comienza con un número).

DOM SET XML ELEMENT VALUE

DOM SET XML ELEMENT VALUE (elementRef {; xRuta}; valorElement {; *})

Parámetro	Tipo		Descripción
elementRef	Cadena	➡	Referencia del elemento XML
xRuta	Texto	➡	Ruta XPath del elemento XML
valorElement	Cadena, Variable	➡	Nuevo valor del elemento
*	Operador	➡	Si se pasa: definir el valor en CDATA

Descripción

El comando DOM SET XML ELEMENT VALUE le permite modificar el valor del elemento definido por refElement.

*Si pasa el parámetro opcional xRuta, usted elige utilizar la notación XPath para indicar el elemento a modificar (para mayor información sobre esta notación, consulte el párrafo "Uso de la notación XPath" en la sección **Presentación de los comandos XML DOM**). En este caso, debe pasar la referencia de un elemento XML raíz en refElement y la ruta XPath del elemento a modificar en xPath.*

En valorElement, pase una cadena o una variable (o un campo) que contenga el nuevo valor del elemento especificado:

- *Si pasa una cadena, el valor se utilizará tal como en la estructura XML.*
- *Si pasa una variable o un campo, 4D procesará el valor, dependiendo del tipo de valorElement.*
  Todos los tipos de datos pueden ser utilizados, a excepción de los arrays, imágenes y punteros.

Cuando se pasa el parámetro opcional asterisco (*), indica que el valor del elemento debe ser definido bajo la forma de CDATA. La forma especial CDATA permite escribir texto sin formato (ver ejemplo 2).

***Nota:** si el elemento designado por refEement es de tipo BLOB, DOM SET XML ELEMENT VALUE lo codifica automáticamente en base64. Sin embargo, el comando **DOM GET XML ELEMENT VALUE** realiza automáticamente la operación inversa.*

Nota sobre el procesamiento de caracteres de fin de línea

Para cumplir con las reglas de procesamiento XML, todas las secuencias de caracteres de fin de línea CR y CRLF se convierten en caracteres LF.

Ejemplo 1

En la siguiente fuente XML:

```
<Book>  <Title>El mejor vendedor</Title> </Book>
```

Si se ejecuta el siguiente código, con vRefElem contiene la referencia del elemento "Titulo":

```
DOM SET XML ELEMENT VALUE(vRefElem;"El Perdedor")
```

Obtenemos:

```
<Book>  <Title>El Perdedor</Title> </Book>
```

Ejemplo 2

En la siguiente fuente XML:

```
<Maths>  <Postulate>1+2=3</Postulate> </Maths>
```

Queremos escribir el texto "12<18" en el elemento <Postulate>. Esta cadena no puede escribirse en XML porque el carácter "<" no se acepta. Este carácter debe transformarse entonces en "<" o debe utilizarse la forma CDATA. Si vElemRef indica el nodo XML <Postulate>:

```
` Forma normal
DOM SET XML ELEMENT VALUE(vElemRef;"12<18")
```

Obtenemos:

```
<Maths>  <Postulate>12 < 18</Postulate> </Maths>
```

```
` CDATA form
DOM SET XML ELEMENT VALUE(vElemRef;"12<18";*)
```

Obtenemos:

```
<Maths>     <Postulate><![CDATA[12 < 18]]></Postulate> </Maths>
```

Variables y conjuntos del sistema

Si el comando ha sido ejecutado correctamente, la variable sistema OK toma el valor 1. Si ocurre un error, toma el valor 0 y se genera un error (por ejemplo, si la referencia del elemento es inválida).

XML SAX

- *Presentación de los comandos XML SAX*
- *SAX ADD PROCESSING INSTRUCTION*
- *SAX ADD XML CDATA*
- *SAX ADD XML COMMENT*
- *SAX ADD XML DOCTYPE*
- *SAX ADD XML ELEMENT VALUE*
- *SAX CLOSE XML ELEMENT*
- *SAX GET XML CDATA*
- *SAX GET XML COMMENT*
- *SAX GET XML DOCUMENT VALUES*
- *SAX GET XML ELEMENT*
- *SAX GET XML ELEMENT VALUE*
- *SAX GET XML ENTITY*
- *SAX Get XML node*
- *SAX GET XML PROCESSING INSTRUCTION*
- *SAX OPEN XML ELEMENT*
- *SAX OPEN XML ELEMENT ARRAYS*
- *SAX SET XML DECLARATION*

Presentación de los comandos XML SAX

Este tema agrupa los comandos XML SAX de 4D.

Para información general sobre XML (presentación, conjunto de caracteres, glosario) como también sobre las diferencias entre los modos DOM y SAX, consulte la sección ***Presentación de los comandos XML DOM****.*

Nota sobre el modo apropiativo*: las referencias XML creadas por un proceso apropiativo sólo se puede utilizar en este proceso específico. Por el contrario, las referencias XML creadas por un proceso cooperativo pueden ser utilizadas por cualquier otro proceso cooperativo, pero no pueden ser utilizadas por cualquier proceso apropiativo.*

Creación, apertura y cierre de documentos XML vía SAX

*Los comandos SAX trabajan con las referencias de documentos estándar de 4D (****DocRef****, referencia de tipo Hora). Por lo tanto es posible utilizar estos comandos de forma conjunta con los comandos 4D utilizados para administrar documentos, tales como* ***SEND PACKET*** *o* ***Append document****.*
La creación y apertura por programación de documentos XML se lleva a cabo utilizando los comandos ***Create document*** *y* ***Open document****. Posteriormente, el uso de un comando XML con estos documentos provocará la implementación automática de los mecanismos XML tales como la codificación. Por ejemplo, el encabezado* `<?xml version="1.0" encoding="… encodage …" standalone = "no "?>` *se escribirá automáticamente en el documento.*

Nota: *los documentos que van a ser leídos por comandos SAX deben ser abiertos en modo sólo lectura por el comando* ***Open document****. Esto evita conflictos entre 4D y la librería Xerces al abrir simultáneamente documentos "estándar" y documentos XML. Si ejecuta un comando de análisis SAX con un documento abierto en modo lectura escritura, se muestra un mensaje de alerta y el análisis es imposible.*

El cierre de un documento XML SAX debe llevarse a cabo utilizando el comando ***CLOSE DOCUMENT****. Si hay elementos XML abiertos, se cerrarán automáticamente.*

SAX ADD PROCESSING INSTRUCTION

SAX ADD PROCESSING INSTRUCTION (documento ; instruccion)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
instruccion	Texto	➡	Instrucción a insertar en el documento

Descripción

*El comando **SAX ADD PROCESSING INSTRUCTION** añade en el documento XML referenciado por documento, una instruccion de procesamiento XML.*

Una instrucción de procesamiento le permite indicar el tipo de aplicación y cuando sea necesario los parámetros adicionales que le permiten procesar una entidad externa no analizable.

El comando da formato a los datos de la instrucción conforme con XML. Sin embargo, las instrucciones misma no son analizadas y depende del desarrollador asegúrarse de que sean válidas.

Ejemplo

El siguiente código:

```
vtInstruccion:="xml-stylesheet type="+Char(Quotes)+"text/xsl"+Char(Quotes)+
"href="+Char(Quotes)+"style.xsl"+Char(Quotes)
SAX ADD PROCESSING INSTRUCTION($DocRef;vtInstruccion)
```

... escribirá la siguiente línea en el documento:

```
<?xml-stylesheet type="text/xsl"href="style.xsl"?>
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

SAX ADD XML CDATA

SAX ADD XML CDATA (documento ; datos)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
datos	BLOB, Texto	➡	Texto o BLOB a insertar en el documento entre etiquetas CData

Descripción

El comando **SAX ADD XML CDATA** añade en el documento XML referenciado por documento, los datos de tipo texto o BLOB. Estos datos se enmarcarán automáticamente con las etiquetas <![CDATA[y]]>
El texto incluido en una sección CData es ignorado por el intérprete XML.

Si quiere codificar los contenidos de datos, debe utilizar el comando **BASE64 ENCODE**. En este caso, por supuesto, debe pasar un BLOB en datos.

Para que este comando funcione correctamente, debe estar abierto un elemento. De lo contrario, se generará un error.

Ejemplo

Si quiere insertar las siguientes líneas en su documento XML:

```
function matchwo(a,b) { if (a < b && a < 0) then { return 1 } else { return 0 } }
```

Para hacer esto, necesita ejecutar el siguiente código:

```
C_TEXT(vtMitexto)
... ` coloque acá el texto en la variable vtMitexto
SAX ADD XML CDATAL($DocRef;vtMitexto)
```

El resultado será:

```
<![CDATA[ function matchwo(a,b) { if (a < b && a < 0) then { return 1 } else { return 0 } } ]]>
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0.

SAX ADD XML COMMENT

SAX ADD XML COMMENT (documento ; comentario)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
comentario	Cadena	➡	Comentario a añadir

Descripción

*El comando **SAX ADD XML COMMENT** añade un comentario en el documento XML referenciado por document.*
Un comentario XML es un texto cuyo contenido no será analizado por el interprete XML. Los comentarios XML deben estar entre los caracteres <!-- y -->.

Ejemplo

La siguiente instrucción:

```
vComentario:="Creado por 4D"
SAX ADD XML COMMENT($DocRef;vComentario)
```

... escribirá la siguiente línea en el documento:

```
<!--Creado por 4D-->
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0.

Gestión de errores

En caso de error, el comando devuelve un error que puede interceptarse utilizando un método de gestión de errores.

SAX ADD XML DOCTYPE

SAX ADD XML DOCTYPE (documento ; docType)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
docType	Cadena	➡	DocType a añadir

Descripción

*El comando **SAX ADD XML DOCTYPE** añade la instrucción DocType definida por el parámetro docType en el documento XML referenciado por document.*

La instrucción DocType permite indicar el tipo de XML en el cual el documento ha sido escrito y especificar la Declaración de tipo de documento (DTD) utilizada. Una instrucción DocType generalmente tiene la siguiente forma: <!DOCTYPE XML_type "DTD_address">.

Ejemplo

La siguiente instrucción:

```
vDocType:="SYSTEM Books \"Book.DTD\""
SAX ADD XML DOCTYPE($DocRef;vDocType)
```

... escribirá la siguiente línea en el documento:

```
<!DOCTYPE SYSTEM Books"Book.DTD">
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0.

Gestión de errores

En caso de error, el comando devuelve un error que puede interceptarse utilizando un método de gestión de errores.

SAX ADD XML ELEMENT VALUE

SAX ADD XML ELEMENT VALUE (documento ; datos {; *})

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
datos	Texto, Variable	➡	Texto o variable a insertar en el documento
*	Operador	➡	Si se pasa: codificar los caracteres especiales Si se omite: no codificación

Descripción

*El comando **SAX ADD XML ELEMENT VALUE** añade directamente en el documento XML referenciado por documento los datos sin convertirlos. Este comando es equivalente, por ejemplo, a insertar un archivo adjunto en el cuerpo de un e-mail.*

En datos, puede pasar directamente una cadena de caracteres, o una variable 4D. El contenido de la variable se convertirá en texto antes de incluirse en el documento XML.

*Si quiere codificar el contenido de datos, debe utilizar el comando **BASE64 ENCODE**. En este caso, por su puesto, debe pasar un BLOB en datos.*

*Por defecto, el comando codifica los caracteres especiales (< > ” ’...) contenidos en los parámetros datos a menos que usted haya desactivado este mecanismo para el proceso actual utilizando el comando **XML SET OPTIONS** pasando el valor XML Raw data a la opción XML String encoding. Por ejemplo:*

```
XML SET OPTIONS($docRef;XML string encoding;XML raw data)
```

*En este contexto, para forzar la codificación de parámetros especiales durante la llamada de **SAX ADD XML ELEMENT VALUE**, debe pasar el parámetro opcional *.*

Para que este comando funcione correctamente, debe estar abierto un elemento. De lo contrario, se generará un error.

Ejemplo

Este ejemplo inserta el archivo whitepaper.pdf en el elemento XML abierto:

```
C_BLOB(vBMiBLOB)
DOCUMENT TO BLOB("c:\\libroblanco.pdf";vBMiBLOB)
SAX ADD XML ELEMENT VALUE($DocRef;vBMiBLOB)
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

SAX CLOSE XML ELEMENT

SAX CLOSE XML ELEMENT (documento)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto

Descripción

*El comando **SAX CLOSE XML ELEMENT** escribe en el documento XML referenciado por documento las instrucciones necesarias para cerrar el último elemento abierto utilizando el comando **SAX OPEN XML**.*

El uso de este comando es opcional. De hecho, 4D añade automáticamente si es necesario, al momento del cierre de los documentos XML, las etiquetas de fin de los elementos no cerrados explícitamente.

Ejemplo

Si el último elemento abierto es <Book>, la siguiente instrucción:

```
SAX CLOSE XML ELEMENT($DocRef)
```

... escribirá la siguiente línea en el documento:

```
</Book>
```

SAX GET XML CDATA

SAX GET XML CDATA (documento ; valor)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
valor	Texto, BLOB	⬅	Valor del elemento

Descripción

El comando **SAX GET XML CDATA** permite recuperar el valor CDATA de un elemento XML existente en el documento XML referenciado por documento. Este comando debe llamarse con el evento SAX XML CDATA. Para mayor información sobre eventos SAX, consulte la descripción del comando **SAX Get XML node**.
Pase una variable valor de tipo Texto si quiere recuperar los datos de tamaño superior a 32 KB (la base debe funcionar en modo Unicode).

Nota de compatibildad: a partir de 4D v12, los contenidos CDATA codificados en base64 son decodificados automáticamente por el comando **SAX GET XML CDATA**, de manera que no es necesario llamar al comando **BASE64 DECODE**.

Ejemplo

Miremos el siguiente código XML:

```
<ElementoRaiz>  <Hijo>Mi Texto<![CDATA[MyCData]]</Hijo> </ElementoRaiz>
```

El siguiente código 4D devolverá "MiCData" en vDatosTexto:

```
C_BLOB(vDator)
C_TEXT(vDatosTexto)
SAX GET XML CDATA(DocRef;vDatos)
vDatosTexto:=BLOB to text(vData;UTF8 C string)
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

SAX GET XML COMMENT

SAX GET XML COMMENT (documento ; comentario)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
comentario	Cadena	⬅	Comentario XML

Descripción

El comando ***SAX GET XML COMMENT*** *devuelve un comentario si un evento SAX de tipo* ***XML Comment*** *se genera en el documento XML referenciado en el parámetro documento. Para mayor información sobre los eventos SAX, consulte la descripción del comando* ***SAX Get XML node****.*

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

SAX GET XML DOCUMENT VALUES

SAX GET XML DOCUMENT VALUES (documento ; codificacion ; version ; autonomo)

Parámetro	Tipo		Descripción
documento	DocRef	⇒	Referencia del documento abierto
codificacion	Cadena	⇐	Conjunto de caracteres del documento XML
version	Cadena	⇐	Versión XML
autonomo	Booleano	⇐	True = el documento es autónomo, de lo contrario es False

Descripción

El comando **SAX GET XML DOCUMENT VALUES** extrae información básica del encabezado XML del documento XML referenciado por documento.

El comando devuelve respectivamente el tipo de codificación, la versión y la propiedad "autónoma" del documento en los parámetros codificacion, version y autonomo. Este comando debe utilizarse en el evento del contexto del evento SAX **XML Start Document**. Para mayor información sobre los eventos SAX, consulte la descripción del comando **SAX Get XML node**.

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

SAX GET XML ELEMENT

SAX GET XML ELEMENT (documento ; nombre ; prefijo ; nomsAtributos ; valoresAtributos)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
nombre	Cadena	⬅	Nombre del elemento
prefijo	Cadena	⬅	Espacio de nombre
nomsAtributos	Array cadena	⬅	Nombres de los atributos
valoresAtributos	Array cadena	⬅	Valores de los atributos

Descripción

*El comando **SAX GET XML ELEMENT** devuelve diversa información relativa al elemento nombre presente en el documento XML referenciado por documento. Este comando debe llamarse con los eventos **XML Start Element** o **XML End Element** SAX. En el caso específico de **XML End Element**, los parámetros de atributos no son manipulados. Para mayor información sobre los eventos SAX, consulte la descripción del comando **SAX Get XML node**.*

El parámetro nombre contiene el nombre del elemento.

El parámetro prefijo devuelve el espacio de nombre (namespace) del elemento. Este parámetro está vacío si ningún espacio de nombre está asociado al elemento.

El comando llena el array nomsAtributos con los nombres de los atributos del elemento objetivo. Si es necesario, el comando crea y dimensiona automáticamente el array.

El comando también llena el array valoresAtributos con los valores de los atributos del elemento objetivo. Si es necesario, el comando crea y dimensiona automáticamente el array.

Ejemplo

Consideremos el siguiente código XML:

```
<ElementoRaiz>
<Hijo Att1="111"Att2="222"Att3="333">MiTexto</Hijo>
</ElementoRaiz>
```

Una vez se ejecuta la siguiente instrucción:

```
SAX GET XML ELEMENT(DocRef;vNombre;vPrefijo;tAttrNombres;tAttrValores)
```

...vNombre contendrá "Hijo"
vPrefix contendrá ""
tAttrNombres{1} contendrá "Att1", tAttrNombres{2} contendrá "Att2", tAttrNombres{3} contendrá "Att3"
tAttrValores{1} contendrá "111", tAttrValores{2} contendrá "222", tAttrValores{3} contendrá "333"

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

SAX GET XML ELEMENT VALUE

SAX GET XML ELEMENT VALUE (documento ; valor)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
valor	Texto, BLOB	⬅	Valor del elemento

Descripción

*El comando **SAX GET XML ELEMENT VALUE** permite recuperar el valor de un elemento XML existente en el documento XML referenciado por documento. Este comando debe llamarse en el contexto de un evento **XML DATA** SAX. Para mayor información sobre los eventos SAX, consulte la descripción del comando **SAX Get XML node**.*

Pase en el parámetro valor una variable de tipo Texto o BLOB. Si pasa un BLOB, el comando automáticamente intentará decodificarlo en base64.

Ejemplo

Miremos el siguiente código XML:

```
<ElementoRaiz>  <Hijo Att1="111" Att2="222" Att3="333">MyText</Hijo> </ElementoRaiz>
```

La siguiente instrucción devolverá "MiText" en vValor:

```
SAX GET XML ELEMENT VALUE(DocRef;vValor)
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

SAX GET XML ENTITY

SAX GET XML ENTITY (documento ; nombre ; valor)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
nombre	Cadena	⬅	Nombre de la entidad
valor	Cadena	⬅	Valor de la entidad

Descripción

*El comando **SAX GET XML ENTITY** permite recuperar el nombre y valor de una entidad XML presente en el documento XML referenciado por documento. Este comando debe llamarse con el evento **XML Entity** SAX. Para mayor información sobre los eventos SAX, consulte la descripción del comando **SAX Get XML node**.*

Ejemplo

Miremos el siguiente código XML:

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE body [  <!ELEMENT body (element*)>  <!ELEMENT element (#PCDATA)>
  <!ENTITY nombre "Remplazo"> ]> <body>  <element>Entidad actualizada por &nombre;</element> </body>
```

La siguiente instrucción devolverá "nombre" en vNom y "Remplazo" en vValor.

```
SAX GET XML ENTITY(DocRef;vNom;vValor)
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

SAX Get XML node

SAX Get XML node (documento) -> Resultado

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
Resultado	Entero largo	⮌	Evento devuelto por la función

Descripción

*El comando **SAX Get XML node** devuelve un entero largo indicando el tipo de evento SAX devuelto durante el análisis del documento XML referenciado por documento.*

*Los eventos que pueden devolverse están disponibles como constantes del tema "**XML**":*

Constante	Tipo	Valor
XML CDATA	Entero largo	7
XML Comment	Entero largo	2
XML DATA	Entero largo	6
XML End Document	Entero largo	9
XML End Element	Entero largo	5
XML Entity	Entero largo	8
XML Processing Instruction	Entero largo	3
XML Start Document	Entero largo	1
XML Start Element	Entero largo	4

Ejemplo

El siguiente ejemplo procesa un evento:

```
DocRef:=Open document("";"xml";Read Mode)
If(OK=1)
   Repeat
      MyEvent:=SAX Get XML node(DocRef)
      Case of
         :(MyEvent=XML Start Document)
            DoSomething
         :(MyEvent=XML Comment)
            DoSomethingElse
      End case
   Until(MyEvent=XML End Document)
   CLOSE DOCUMENT(DocRef)
End if
```

Variables y conjuntos del sistema

Si el comando se ejecuta correctamente, la variable sistema OK toma el valor 1. De lo contrario toma el valor 0 y se genera un error.

SAX GET XML PROCESSING INSTRUCTION

SAX GET XML PROCESSING INSTRUCTION (documento ; nombre ; valor)

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
nombre	Cadena	⬅	Nombre de la instrucción
valor	Cadena	⬅	Valor de la instrucción

Descripción

*El comando **SAX GET XML PROCESSING INSTRUCTION** devuelve el nombre y el valor de la instrucción de proceso XML analizada en el documento XML referenciada por documento. Este comando debe llamarse con el evento **XML Processing Instruction**. Para mayor información sobre eventos SAX, consulte la descripción del comando **SAX Get XML node**.*

Ejemplo

Miremos el siguiente código XML:

```
<?xml version="1.0" encoding="UTF-8"?> <!-- Edited with XML Spy v3.0.7 NT (http://www.xmlspy.com) by Myself (4D SA)--> <?PI TextProcess?> <!DOCTYPE RootElement SYSTEM "ParseTest.dtd">
```

La siguiente instrucción devolverá "PI" en vNom y "TextProcess" en vValor:

```
SAX GET XML PROCESSING INSTRUCTION($DocRef;vNom;vValor)
```

SAX OPEN XML ELEMENT

SAX OPEN XML ELEMENT (documento ; etiqueta {; nomAtrib ; valorAtrib} {; nomAtrib2 ; valorAtrib2 ; ... ; nomAtribN ; valorAtribN})

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
etiqueta	Cadena	➡	Nombre del elemento a abrir
nomAtrib	Cadena	➡	Nombre del atributo
valorAtrib	Cadena	➡	Valor del atributo

Descripción

*El comando **SAX OPEN XML ELEMENT** permite añadir un nuevo elemento en el documento XML referenciado por document como también, opcionalmente, atributos y sus valores.*

El elemento añadido está "abierto" en el documento (la etiqueta de fin no está añadida). Para cerrar un elemento creado utilizando este comando, debe:

- *Utilizar el comando **SAX CLOSE XML ELEMENT**, o*
- *Cerrar el documento XML. En este caso, 4D añadirá automáticamente las etiquetas XML de cierre necesarias.*

En etiqueta, pase el nombre del elemento a crear. Este nombre sólo puede contener letras, números y los caracteres ".", "-","_" y ":". Si se pasa un caracter inválido en etiqueta, se generará un error.

Opcionalmente, el comando permite pasar uno o varios pares de atributos/valores (en forma de variables, campos o valores literales) utilizando los parámetros nomAtrib y valorAtrib. Puede pasar tantos pares atributo/valor como quiera.

Ejemplo

La siguiente instrucción:

```
vElement:="Libro"
SAX OPEN XML ELEMENT($DocRef;vElement)
```

... escribirá la siguiente línea en el documento:

```
<Book
```

Gestión de errores

Si se pasa en tag un carácter inválido, se genera un error.

SAX OPEN XML ELEMENT ARRAYS

SAX OPEN XML ELEMENT ARRAYS (documento ; etiqueta {; arrayNomsAtrib ; arrayValoresAtrib} {; arrayNomsAtrib2 ; arrayValoresAtrib2 ; ... ; arrayNomsAtribN ; arrayValoresAtribN})

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
etiqueta	Cadena	➡	Nombre del elemento a abrir
arrayNomsAtrib	Array cadena	➡	Array de nombres de atributos
arrayValoresAtrib	Array cadena, Array entero largo, Array fecha, Array real, Array imagen, Array booleano	➡	Array de valores de atributos

Descripción

*El comando **SAX OPEN XML ELEMENT ARRAYS** permite añadir un nuevo elemento en el documento XML cuya referencia se pasa en document como también, opcionalmente, atributos y sus valores en forma de arrays.*

*Excepto por el soporte de arrays (ver a continuación), este comando es idéntico a **SAX OPEN XML ELEMENT**. Por favor consulte la descripción de este comando para mayor información sobre su funcionamiento.*

***SAX OPEN XML ELEMENT ARRAYS** acepta arrays de tipo fecha, numéricos, booleanos e imagen como parámetros arrayValoresAtrib. 4D automáticamente realiza todas las conversiones necesarias; puede configurar estas conversiones utilizando el comando **XML SET OPTIONS**.*

*Opcionalmente, el comando **SAX OPEN XML ELEMENT ARRAYS** permite pasar pares de atributos y de valores de atributos en forma de arrays en los parámetros arrayNomsAtrib y arrayValoresAtrib.*

Los arrays deben haber sido creados previamente y funcionar por pares. Puede pasar tantos pares de arrays y elementos en cada par como quiera.

Ejemplo

El siguiente método:

```
ARRAY STRING(80;aNomsAtrib;2)
ARRAY STRING(80;aValoresAtrib;2)
vElement:="Libro"
aNomsAtrib{1}:="Font"
aValoresAtrib{1}:="Arial"
aNomsAtrib{2}:="Style"
aValoresAtrib{2}:="Bold"
SAX OPEN XML ELEMENT ARRAYS($DocRef;vElemento;tNomsAtrib;tValoresAtrib)
```

... escribirá en el documento:

```
<Book Font="Arial" Style="Bold">
```

SAX SET XML DECLARATION

SAX SET XML DECLARATION (documento ; codificacion {; autonomo {; indentacion}})

Parámetro	Tipo		Descripción
documento	DocRef	➡	Referencia del documento abierto
codificacion	Cadena	➡	Conjunto de caracteres del documento XML
autonomo	Booleano	➡	True = el documento es autónomo False (por defecto) = el documento no es autónomo
indentacion	Booleano	➡	*** Obsoleto no utilizar ***

Descripción

El comando SAX SET XML DECLARATION inicializa el documento XML referenciado por documento utilizando los valores pasados en el parámetro. Estos parámetros permiten determinar la codificación, el atributo autónomo y la indentación del documento.

- codificacion: indica el conjunto de caracteres utilizado en el documento. Por defecto (si no se llama al comando), se utiliza el conjunto de caracteres UTF-8 (Unicode comprimido).
  Nota: si pasa un conjunto de caracteres que no soporta los comandos XML de 4D, se utilizará UTF-8. Consulte **Conjuntos de caracteres** para ver la lista de conjuntos de caracteres soportados (sin embargo se recomienda UTF-8 en la mayoría de los casos).

- autonomo: indica si el documento es autonomo (**True**) o si depende, para su funcionamiento, de otros archivos o de recursos externos (**False**). Por defecto (si no se llama el comando o si el parámetro se omite), el documento no es autónomo.

Este comando debe llamarse una sola vez por documento y antes del primer comando de escritura XML en el documento; de lo contrario, se generará un mensaje de error.

Nota de compatibilidad: el parámetro indentacion se conserva por razones de compatibilidad con versiones anteriores de 4D pero su uso no se recomienda a partir de 4D v12. De ahora en adelante, para definir la indentación del documento, se recomienda utilizar el comando **XML SET OPTIONS**.

Este comando debe llamarse una sola vez por documento y antes del primer comando de escritura XML en el documento; de lo contrario, se generará un mensaje de error.

Ejemplo

El siguiente código:

```
SAX SET XML DECLARATION($DocRef;"UTF-16";True)
```

... escribirá esta línea en el documento:

```
<?xml version="1.0"Encoding="UTF-16"standalone="sí"?>
```

Lista de temas de constantes

4D Write Pro
Abrir ventana formulario
Acceso objetos diseño
Acción estándar
Área de formulario
Atajos de teclado
Atributos de texto multiestilo
Backup
BLOB
Buscar ventana
Cadenas
Caracteres latinos ISO
Carpeta sistema
Códigos ASCII
Colores
Compresión de imagen
Comunicaciones
Crear ventana
DEFINIR COLORES RVA
Destinos de búsqueda
Días y meses
Diccionarios
Documentos sistema
Entorno 4D
Estado del proceso
Estilos de fuente
Euro monedas
Eventos (Modificadores)
Eventos (tipos)
Eventos de la base
Eventos trigger
Expresiones
Firmas del sistema estándar
Form event
Formato de sistema
Formatos de salida de fechas
Formatos de salida de hora
Formatos de salida de imágenes
Fórmulas
Funciones matemáticas
Gestión
Historial de eventos
HTTP Client
Interfaz de la plataforma
LDAP
Licencia disponible
List box cálculo pie
Listas jerárquicas
Listbox
Mantenimiento archivo de datos
Motor de la base
Nombres de metadatos imágenes
Números de puerto TCP
Objetos de formulario (Acceso)
Objetos y colecciones
Opciones de impresión
Parámetro de formulario
Parámetros de la base
Parámetros Gráficos
PHP
Picture Metadata Values
Portapapeles
Profundidad de la pantalla
Propiedades de ítem de menú
Propiedades de la plataforma
Propiedades de los objetos
Propiedades de los recursos
QR Bordes
QR Comandos
QR Destino de salida
QR Filas para propiedades
QR Operadores
QR Propiedades de área
QR Propiedades de documento
QR Propiedades de texto
QR Tipos de informes

- *Relaciones*
- *Servicios Web (Cliente)*
- *Servicios Web (Servidor)*
- *Servidor web*
- *SQL*
- *Teclas de función*
- *Texto multiestilo*
- *Tipo de índice*
- *Tipo de lista de las fuentes*
- *Tipo de proceso*
- *Tipo Digest*
- *Tipos de campos y variables*
- *Tipos objetos formulario*
- *Transformación de imágenes*
- *Valores para acción estándar asociada*
- *Ventana*
- *Web Area*
- *XML*

4D Write Pro

Constante	Tipo	Valor	Comentario
wk 4wp	Entero largo	4	El documento 4D Write Pro se guarda en un formato de archivo nativo (HTML comprimido con imágenes almacenadas en una carpeta separada). Las expresiones 4D no se calculan y las etiquetas 4D especificas se incluyen. Este formato es especialmente adecuado para guardar y archivar documentos 4D Write Pro en el disco sin ninguna pérdida.
wk armenian	Entero largo	19	
wk author	Cadena	author	
wk auto	Entero largo	0	
wk background clip	Cadena	backgroundClip	
wk background color	Cadena	backgroundColor	
wk background image	Cadena	backgroundImage	
wk background origin	Cadena	backgroundOrigin	
wk background position h	Cadena	backgroundPositionH	
wk background position v	Cadena	backgroundPositionV	
wk background repeat	Cadena	backgroundRepeat	
wk background size h	Cadena	backgroundSizeH	
wk background size v	Cadena	backgroundSizeV	
wk bar	Entero largo	4	
wk baseline	Entero largo	4	
wk border box	Entero largo	0	
wk border color	Cadena	borderColor	
wk border color bottom	Cadena	borderColorBottom	
wk border color left	Cadena	borderColorLeft	
wk border color right	Cadena	borderColorRight	
wk border color top	Cadena	borderColorTop	
wk border radius	Cadena	borderRadius	
wk border style	Cadena	borderStyle	
wk border style bottom	Cadena	borderStyleBottom	
wk border style left	Cadena	borderStyleLeft	
wk border style right	Cadena	borderStyleRight	
wk border style top	Cadena	borderStyleTop	
wk border width	Cadena	borderWidth	
wk border width bottom	Cadena	borderWidthBottom	
wk border width left	Cadena	borderWidthLeft	

Constante	Tipo	Valor	Comentario
wk border width right	Cadena	borderWidthRight	
wk border width top	Cadena	borderWidthTop	
wk bottom	Entero largo	1	
wk capitalize	Entero largo	1	
wk center	Entero largo	2	
wk circle	Entero largo	11	
wk cjk ideographic	Entero largo	24	
wk club	Entero largo	27	
wk company	Cadena	company	
wk contain	Entero largo	-1	
wk content box	Entero largo	2	
wk cover	Entero largo	-2	
wk custom	Entero largo	29	
wk dashed	Entero largo	3	
wk date creation	Cadena	dateCreation	
wk date modified	Cadena	dateModified	
wk decimal	Entero largo	3	
wk decimal greek	Entero largo	28	
wk decimal leading zero	Entero largo	13	
wk default	Entero largo	-1	
wk diamond	Entero largo	26	
wk direction	Cadena	direction	
wk disc	Entero largo	10	
wk dotted	Entero largo	2	
wk double	Entero largo	4	
wk dpi	Cadena	dpi	
wk end text	Entero largo	0	
wk false	Entero largo	0	
wk font	Cadena	font	
wk font bold	Cadena	fontBold	
wk font family	Cadena	fontFamily	
wk font italic	Cadena	fontItalic	
wk font size	Cadena	fontSize	
wk georgian	Entero largo	20	
wk groove	Entero largo	6	
wk hebrew	Entero largo	21	
wk height	Cadena	height	

Constante	Tipo	Valor	Comentario
wk hidden	Entero largo	5	
wk hiragana	Entero largo	22	
wk hollow square	Entero largo	25	
wk html debug	Entero largo	1	Código HTML formateado ("pretty print"), más fácil de depurar
wk image	Cadena	image	
wk image alternative text	Cadena	imageAltText	
wk inset	Entero largo	8	
wk inside	Cadena	Inside	
wk justify	Entero largo	5	
wk katakana	Entero largo	23	
wk layout unit	Cadena	userUnit	
wk left	Entero largo	0	
wk left to right	Entero largo	0	
wk line height	Cadena	lineHeight	
wk linethrough	Entero largo	2	
wk list auto	Entero largo	2147483647	
wk list font	Cadena	listFont	
wk list font family	Cadena	listFontFamily	
wk list start number	Cadena	listStartNumber	
wk list string format LTR	Cadena	listStringFormatLtr	
wk list string format RTL	Cadena	listStringFormatRtl	
wk list style image	Cadena	listStyleImage	
wk list style image height	Cadena	listStyleImageHeight	
wk list style type	Cadena	listStyleType	
wk lower greek	Entero largo	18	
wk lower latin	Entero largo	14	
wk lower roman	Entero largo	15	
wk lowercase	Entero largo	2	
wk margin	Cadena	margin	
wk margin bottom	Cadena	marginBottom	
wk margin left	Cadena	marginLeft	
wk margin right	Cadena	marginRight	
wk margin top	Cadena	marginTop	
wk middle	Entero largo	2	
wk mime html	Entero largo	1	El documento 4D Write Pro se guarda como MIME HTML con documentos los documentos html y las imágenes embebidas como partes MIME (codificado en base 64). Las expresiones se calculan y las etiquetas 4D específicas se eliminan. Este formato es especialmente adecuado para el envío de mensajes de correo electrónico HTML con el comando **SMTP_QuickSend**.

Constante	Tipo	Valor	Comentario
wk min height	Cadena	minHeight	
wk min width	Cadena	minWidth	
wk mixed	Entero largo	-2147483648	
wk new line style sheet	Cadena	newLineStyleSheet	
wk no repeat	Entero largo	3	
wk none	Entero largo	0	
wk normal	Entero largo	0	Código HTML estándar
wk notes	Cadena	notes	
wk outset	Entero largo	9	
wk outside	Cadena	Outside	
wk padding	Cadena	padding	
wk padding bottom	Cadena	paddingBottom	
wk padding box	Entero largo	1	
wk padding left	Cadena	paddingLeft	
wk padding right	Cadena	paddingRight	
wk padding top	Cadena	paddingTop	
wk range end	Cadena	rangeEnd	
wk range owner	Cadena	rangeOwner	
wk range start	Cadena	rangeStart	
wk repeat	Entero largo	0	
wk repeat x	Entero largo	1	
wk repeat y	Entero largo	2	
wk ridge	Entero largo	7	
wk right	Entero largo	1	
wk right to left	Entero largo	1	
wk semi transparent	Entero largo	5	
wk small uppercase	Entero largo	4	
wk solid	Entero largo	1	
wk square	Entero largo	12	
wk start text	Entero largo	1	
wk style sheet	Cadena	styleSheet	
wk subject	Cadena	subject	
wk subscript	Entero largo	6	
wk superscript	Entero largo	5	
wk tab stop offsets	Cadena	tabStopOffsets	
wk tab stop types	Cadena	tabStopTypes	
wk text align	Cadena	textAlign	
wk text color	Cadena	color	

Constante	Tipo	Valor	Comentario
wk text indent	Cadena	textIndent	
wk text linethrough color	Cadena	textLinethroughColor	
wk text linethrough style	Cadena	textLinethroughStyle	
wk text shadow color	Cadena	textShadowColor	
wk text shadow offset	Cadena	textShadowOffset	
wk text transform	Cadena	textTransform	
wk text underline color	Cadena	textUnderlineColor	
wk text underline style	Cadena	textUnderlineStyle	
wk title	Cadena	title	
wk top	Entero largo	0	
wk transparent	Entero largo	-1	
wk true	Entero largo	1	
wk underline	Entero largo	1	
wk unit cm	Cadena	cm	
wk unit inch	Cadena	in	
wk unit mm	Cadena	mm	
wk unit percent	Cadena	%	
wk unit pt	Cadena	pt	
wk unit px	Cadena	px	
wk upper latin	Entero largo	16	
wk upper roman	Entero largo	17	
wk uppercase	Entero largo	3	
wk value unit not percentage	Entero largo	-100000	
wk value unit percentage	Entero largo	-100001	
wk version	Cadena	version	
wk vertical align	Cadena	verticalAlign	
wk web page complete	Entero largo	2	Extensión .htm o .html. El documento se guarda como HTML estándar y sus recursos se guardan por separado. Las etiquetas 4D específicas se eliminan y las expresiones se calculan. Este formato es especialmente adecuado cuando se quiere mostrar un documento 4D Write Pro en un navegador web.
wk web page html 4D	Entero largo	3	El documento 4D Write Pro se guarda como HTML e incluye las etiquetas 4D específicas; cada expresión se inserta como un espacio de no separación. Dado que este formato es sin pérdidas, es apropiado para el almacenamiento en un campo texto.
wk width	Cadena	width	
wk word	Entero largo	6	

*Para mayor información, por favor consulte la sección **Lenguaje 4D Write Pro**.*

Constante	Tipo	Valor	Comentario

Abrir ventana formulario

Constante	Tipo	Valor	Comentario
_o_Compositing mode form	Entero largo	4096	*** Constante obsoleta ***
_o_Has toolbar button Mac OS	Entero largo	8192	*** Constante obsoleta ***
At the bottom	Entero largo	393216	
At the top	Entero largo	327680	
Controller form window	Entero largo	133056	
Form has full screen mode Mac	Entero largo	65536	
Form has no menu bar	Entero largo	2048	
Horizontally centered	Entero largo	65536	
Modal form dialog box	Entero largo	1	
Movable form dialog box	Entero largo	5	
On the left	Entero largo	131072	
On the right	Entero largo	196608	
Palette form window	Entero largo	1984	
Plain form window	Entero largo	8	
Pop up form window	Entero largo	32	
Sheet form window	Entero largo	33	
Toolbar form window	Entero largo	35	
Vertically centered	Entero largo	262144	

Acceso objetos diseño

Constante	Tipo	Valor	Comentario
Attribute executed on server	Entero largo	8	Corresponds to the "Execute on server" option
Attribute folder name	Entero largo	1024	Nombre de la carpeta para el método (atributo "carpeta"). Cuando pase esta constante, debe pasar un nombre de carpeta en attribValue: • si el nombre corresponde a una carpeta válida, el método se coloca en esta carpeta padre, • si la carpeta no existe, el comando no cambia nada en el nivel de la carpeta padre, • si pasa una cadena vacía, el método se ubica al nivel de la raíz.
Attribute invisible	Entero largo	1	Corresponde a la opción "Invisible"
Attribute published SOAP	Entero largo	3	Corresponde a la opción "Ofrecido como servicio web"
Attribute published SQL	Entero largo	7	Corresponde a la opción "Disponible vía SQL"
Attribute published Web	Entero largo	2	Corresponde a la opción "Disponible vía las etiquetas HTML y los URLs 4D (4DACTION...)"
Attribute published WSDL	Entero largo	4	Corresponde a la opción "Publicado en WSDL"
Attribute shared	Entero largo	5	Corresponde a la opción "Compartido entre componentes y base local"
Code with tokens	Entero largo	1	Incluir los tokens en el código exportado
On object locked abort	Entero largo	0	La carga del objeto se aborta (funcionamiento por defecto)
On object locked confirm	Entero largo	2	4D muestra una caja de diálogo permitiéndole elegir entre intentar nuevamente o abortar. En modo remoto, esta opción no es soportada (la carga se abandona)
On object locked retry	Entero largo	1	4D intenta cargar el objeto hasta que sea liberado
Path all objects	Entero largo	31	Combinación de las rutas de todos los métodos de la base
Path database method	Entero largo	2	Path of database methods specified (English name). List of these methods: [databaseMethod]/onStartup [databaseMethod]/onExit [databaseMethod]/onDrop [databaseMethod]/onBackupStartup [databaseMethod]/onBackupShutdown [databaseMethod]/onWebConnection [databaseMethod]/onWebAuthentication [databaseMethod]/onWebSessionSuspend [databaseMethod]/onServerStartup [databaseMethod]/onServerShutdown [databaseMethod]/onServerOpenConnexion [databaseMethod]/onServerCloseConnection [databaseMethod]/onSystemEvent [databaseMethod]/onSqlAuthentication
Path project form	Entero largo	4	Ruta de los métodos formulario proyecto y de todos su métodos objeto. Ejemplos: [projectForm]/myForm/{formMethod} [projectForm]/myForm/button1 [projectForm]/myForm/my%2list [projectForm]/myForm/button1
Path project method	Entero largo	1	Nombre del método. Ejemplo: MiMetodoProyecto
Path table form	Entero largo	16	Ruta de los métodos formulario tabla y de todos sus métodos objeto. Ejemplos: [tableForm]/table_1/Form1/{formMethod} [tableForm]/table_1/Form1/button1 [tableForm]/table_1/Form1/my%2list [tableForm]/table_2/Form1/my%2list
Path trigger	Entero largo	8	Ruta de los triggers de la base. Ejemplos: [trigger]/tabla_1 [trigger]/tabla_2

Acción estándar

Constante	Tipo	Valor	Comentario
_o_Object Accept action	Cadena	2	
_o_Object Add subrecord action	Cadena	14	
_o_Object Automatic splitter action	Cadena	16	
_o_Object Cancel action	Cadena	1	
_o_Object Clear action	Cadena	21	
_o_Object Copy action	Cadena	19	
_o_Object Cut action	Cadena	18	
_o_Object Database Settings action	Cadena	32	
_o_Object Delete record action	Cadena	7	
_o_Object Delete subrecord action	Cadena	13	
_o_Object Edit subrecord action	Cadena	12	
_o_Object First page action	Cadena	10	
_o_Object First record action	Cadena	5	
_o_Object Goto page action	Cadena	15	
_o_Object Last page action	Cadena	11	
_o_Object Last record action	Cadena	6	
_o_Object MSC action	Cadena	36	
_o_Object Next page action	Cadena	8	
_o_Object Next record action	Cadena	3	
_o_Object No standard action	Cadena	0	
_o_Object Open back URL action	Cadena	37	
_o_Object Open next URL action	Cadena	38	
_o_Object Paste action	Cadena	20	
_o_Object Previous page action	Cadena	9	
_o_Object Previous record action	Cadena	4	
_o_Object Quit action	Cadena	27	
_o_Object Redo action	Cadena	31	
_o_Object Refresh current URL action	Cadena	39	
_o_Object Return to Design mode action	Cadena	35	
_o_Object Select all action	Cadena	22	
_o_Object Show Clipboard action	Cadena	23	
_o_Object Stop loading URL action	Cadena	40	
_o_Object Test Application action	Cadena	26	

Constante	Tipo	Valor	Comentario
_o_Object Undo action	Cadena	17	
ak accept	Cadena	accept	
ak add subrecord	Cadena	addSubrecord	
ak automatic splitter	Cadena	automaticSplitter	
ak background color	Cadena	backgroundColor	Displays the standard background color submenu
ak background color dialog	Cadena	backgroundColor/showDialog	Opens font background color dialog
ak cancel	Cadena	cancel	
ak clear	Cadena	clear	El objetivo de esta acción es siempre el objeto que tiene el foco del teclado
ak compute expressions	Cadena	computeExpressions	Actualiza todas las expresiones dinámicas en el área
ak copy	Cadena	copy	El objetivo de esta acción es siempre el objeto que tiene el foco del teclado
ak current form	Entero largo	1	El formulario actual es el formulario donde se llamó la acción. Podría ser el formulario principal o un formulario tipo paleta delante del formulario principal del proceso actual.
ak cut	Cadena	cut	El objetivo de esta acción es siempre el objeto que tiene el foco del teclado
ak database settings	Cadena	databaseSettings	Muestra el diálogo de Configuración de la base de datos estándar.
ak delete record	Cadena	deleteRecord	
ak delete subrecord	Cadena	deleteSubrecord	
ak display subrecord	Cadena	displaySubrecord	
ak edit subrecord	Cadena	editSubrecord	
ak first page	Cadena	firstPage	
ak first record	Cadena	firstRecord	
ak font bold	Cadena	fontBold	Alterna el atributo de fuente en negrita
ak font color	Cadena	color	Font color attribute
ak font color dialog	Cadena	color/showDialog	Muestra el diálogo del color de fuente del sistema
ak font italic	Cadena	fontItalic	Activa el atributo de fuente en cursiva
ak font linethrough	Cadena	fontLinethrough	Activa el atributo de fuente tachado
ak font show dialog	Cadena	font/showDialog	Muestra el diálogo del selector de fuente del sistema
ak font size	Cadena	fontSize	Font size attribute
ak font style	Cadena	fontStyle	Muestra el submenú de estilo de fuente estándar
ak font underline	Cadena	fontUnderline	Activa el atributo de fuente de subrayado
ak freeze expressions	Cadena	freezeExpressions	Congela todas las expresiones dinámicas en el área
ak goto page	Cadena	gotoPage	parámetro: "?value=pageNumber"
ak last page	Cadena	lastPage	
ak last record	Cadena	lastRecord	
ak main form	Entero largo	2	El formulario principal es el documento más adelante o el formulario diálogo del proceso, excluyendo cualquier ventana flotante o emergente.
ak msc	Cadena	msc	Muestra la ventana **Centro de seguridad y mantenimiento**.
ak next page	Cadena	nextPage	
ak next record	Cadena	nextRecord	
ak none	Cadena	""	
ak open back url	Cadena	openBackURL	Abre la URL anterior en la secuencia de navegación realizada por el usuario en el área Web.
ak open forward url	Cadena	openForwardURL	Abre la siguiente URL en la secuencia de navegación realizada por el usuario en el área Web.
ak paste	Cadena	paste	El objetivo de esta acción es siempre el objeto que tiene el foco del teclado
ak previous page	Cadena	previousPage	
ak previous record	Cadena	previousRecord	
ak quit	Cadena	quit	Muestra un diálogo de confirmación "¿Está seguro?", a continuación, luego sale de la aplicación 4D si se confirma.
ak redo	Cadena	redo	El objetivo de esta acción es siempre el objeto que tiene el foco del teclado
ak refresh current url	Cadena	refreshCurrentURL	Recarga el contenido actual del área Web.

Constante	Tipo	Valor	Comentario
ak return to design mode	*Cadena*	*design*	*Trae las ventanas y las barras de menú del entorno Diseño 4D al primer plano.*
ak select all	*Cadena*	*selectAll*	*El objetivo de esta acción es siempre el objeto que tiene el foco del teclado*
ak show clipboard	*Cadena*	*showClipboard*	
ak show reference	*Cadena*	*visibleReferences*	*Muestra todas las expresiones dinámicas como referencias*
ak standard action title	*Cadena*	*4D_action_title*	*Título localizado por defecto para la acción estándar. Incluye el nombre de la acción localizada e información adicional si es pertinente, por ejemplo "Deshacer <acción anterior>".*
ak stop loading url	*Cadena*	*stopLoadingURL*	*Detiene la carga de la página y/u objetos de la URL actual en el área Web.*
ak undo	*Cadena*	*undo*	*El objetivo de esta acción es siempre el objeto que tiene el foco del teclado*

Área de formulario

Constante	*Tipo*	*Valor*	*Comentario*
Form break0	*Entero largo*	*300*	
Form break1	*Entero largo*	*301*	
Form break2	*Entero largo*	*302*	
Form break3	*Entero largo*	*303*	
Form break4	*Entero largo*	*304*	
Form break5	*Entero largo*	*305*	
Form break6	*Entero largo*	*306*	
Form break7	*Entero largo*	*307*	
Form break8	*Entero largo*	*308*	
Form break9	*Entero largo*	*309*	
Form detail	*Entero largo*	*0*	
Form footer	*Entero largo*	*100*	
Form header	*Entero largo*	*200*	
Form header1	*Entero largo*	*201*	
Form header10	*Entero largo*	*210*	
Form header2	*Entero largo*	*202*	
Form header3	*Entero largo*	*203*	
Form header4	*Entero largo*	*204*	
Form header5	*Entero largo*	*205*	
Form header6	*Entero largo*	*206*	
Form header7	*Entero largo*	*207*	
Form header8	*Entero largo*	*208*	
Form header9	*Entero largo*	*209*	

Atajos de teclado

Constante	*Tipo*	*Valor*	*Comentario*
Shortcut with Backspace	*Cadena*	*[backspace]*	
Shortcut with Carriage Return	*Cadena*	*[return]*	
Shortcut with Delete	*Cadena*	*[del]*	
Shortcut with Down arrow	*Cadena*	*[down arrow]*	
Shortcut with End	*Cadena*	*[end]*	
Shortcut with Enter	*Cadena*	*[enter]*	
Shortcut with Escape	*Cadena*	*[esc]*	
Shortcut with F1	*Cadena*	*[F1]*	
Shortcut with F10	*Cadena*	*[F10]*	
Shortcut with F11	*Cadena*	*[F11]*	
Shortcut with F12	*Cadena*	*[F12]*	
Shortcut with F13	*Cadena*	*[F13]*	
Shortcut with F14	*Cadena*	*[F14]*	
Shortcut with F15	*Cadena*	*[F15]*	
Shortcut with F2	*Cadena*	*[F2]*	
Shortcut with F3	*Cadena*	*[F3]*	
Shortcut with F4	*Cadena*	*[F4]*	
Shortcut with F5	*Cadena*	*[F5]*	
Shortcut with F6	*Cadena*	*[F6]*	
Shortcut with F7	*Cadena*	*[F7]*	
Shortcut with F8	*Cadena*	*[F8]*	
Shortcut with F9	*Cadena*	*[F9]*	
Shortcut with Help	*Cadena*	*[help]*	
Shortcut with Home	*Cadena*	*[home]*	
Shortcut with Left arrow	*Cadena*	*[left arrow]*	
Shortcut with Page down	*Cadena*	*[page down]*	
Shortcut with Page up	*Cadena*	*[page up]*	
Shortcut with Right arrow	*Cadena*	*[right arrow]*	
Shortcut with Tabulation	*Cadena*	*[tab]*	
Shortcut with Up arrow	*Cadena*	*[up arrow]*	

Atributos de texto multiestilo

Constante	*Tipo*	*Valor*	*Comentario*
Attribute background color	*Entero largo*	*8*	*attValue=Valor hexadecimal o nombre del color HTML (Windows únicamente)*
Attribute bold style	*Entero largo*	*1*	*attValue=0: elimina el atributo negrita de la selección* *attValue=1: aplica el atributo negrita a la selección*
Attribute font name	*Entero largo*	*5*	*attValue=nombre de la familia de la fuente (cadena)*
Attribute italic style	*Entero largo*	*2*	*attValue=0: elimina el atributo itálica de la selección* *attValue=1: aplica el atributo itálica a la selección.*
Attribute strikethrough style	*Entero largo*	*3*	*attValue=0: elimina el atributo tachado de la selección* *attValue=1: aplica el atributo tachado a la selección*
Attribute text color	*Entero largo*	*7*	*attValue=valores hexadecimales o nombre de color HML*
Attribute text size	*Entero largo*	*6*	*attValue=número de puntos(número)*
Attribute underline style	*Entero largo*	*4*	*attValue=0: elimina el atributo subrayado de la selección* *attValue=1: aplica el atributo subrayado a la selección*

Backup

Constante	Tipo	Valor	Comentario
Auto repair mode	Entero largo	1	Utilizar el modo flexible con las acciones de reparación automática y llenar el parámetro errObject (si lo hay)
Field attribute with name	Entero largo	2	Los campos son identificados por su nombre. Ejemplo: {"Apellido":"Gómez"}
Field attribute with number	Entero largo	1	Los campos se identifican por su número (por defecto si se omite). Ejemplo: { "5":"Jones"}.
Last backup date	Entero largo	0	
Last backup status	Entero largo	2	
Last restore date	Entero largo	0	
Last restore status	Entero largo	2	
Next backup date	Entero largo	4	
Strict mode	Entero largo	0	Utilice el modo de integración estricto (por defecto)

BLOB

Constante	Tipo	Valor	Comentario
Compact compression mode	Entero largo	1	Compresión interna más compacta (en detrimento de la velocidad a la cual la compresión y descompresión se efectúan). Método por defecto.
Extended real format	Entero largo	1	
Fast compression mode	Entero largo	2	Compresión más rápida en detrimento (y será descomprimido lo más rápido posible), en detrimento de la tasa de compresión (una vez comprimido, el BLOB será más grande).
GZIP best compression mode	Entero largo	-1	Compresión GZIP más compacta (en detrimento de la velocidad a la cual la compresión y decompresión se efectúan).
GZIP fast compression mode	Entero largo	-2	Compresión/descompresión GZIP más rápida (en detrimento de la tasa de compresión).
Is not compressed	Entero largo	0	Sin compresión
Mac C string	Entero largo	0	
Mac Pascal string	Entero largo	1	
Mac text with length	Entero largo	2	
Mac text without length	Entero largo	3	
Macintosh byte ordering	Entero largo	1	
Macintosh double real format	Entero largo	2	
Native byte ordering	Entero largo	0	
Native real format	Entero largo	0	
PC byte ordering	Entero largo	2	
PC double real format	Entero largo	3	
UTF8 C string	Entero largo	4	
UTF8 text with length	Entero largo	5	
UTF8 text without length	Entero largo	6	

Buscar ventana

Constante	Tipo	Valor	Comentario
_o_In contents	Entero largo	3	Plataforma: Mac OS y Windows

Cadenas

Constante	Tipo	Valor	Comentario
sk ignore empty strings	Entero largo	1	Remove empty strings from the resulting collection (they are ignored)
sk trim spaces	Entero largo	2	Trim space characters at the beginning and end of substrings

Caracteres latinos ISO

Constante	Tipo	Valor	Comentario
ISO L1 a acute	Cadena	á	
ISO L1 a circumflex	Cadena	â	
ISO L1 a grave	Cadena	à	
ISO L1 a ring	Cadena	å	
ISO L1 a tilde	Cadena	ã	
ISO L1 a umlaut	Cadena	ä	
ISO L1 ae ligature	Cadena	æ	
ISO L1 Ampersand	Cadena	&	
ISO L1 c cedilla	Cadena	ç	
ISO L1 Cap A acute	Cadena	Á	
ISO L1 Cap A circumflex	Cadena	Â	
ISO L1 Cap A grave	Cadena	À	
ISO L1 Cap A ring	Cadena	Å	
ISO L1 Cap A tilde	Cadena	Ã	
ISO L1 Cap A umlaut	Cadena	Ä	
ISO L1 Cap AE ligature	Cadena	&AELig;	
ISO L1 Cap C cedilla	Cadena	Ç	
ISO L1 Cap E acute	Cadena	É	
ISO L1 Cap E circumflex	Cadena	Ê	
ISO L1 Cap E grave	Cadena	È	
ISO L1 Cap E umlaut	Cadena	Ë	
ISO L1 Cap Eth Icelandic	Cadena	Ð	
ISO L1 Cap I acute	Cadena	Í	
ISO L1 Cap I circumflex	Cadena	Î	
ISO L1 Cap I grave	Cadena	Ì	
ISO L1 Cap I umlaut	Cadena	Ï	
ISO L1 Cap N tilde	Cadena	Ñ	
ISO L1 Cap O acute	Cadena	Ó	
ISO L1 Cap O circumflex	Cadena	Ô	
ISO L1 Cap O grave	Cadena	Ò	
ISO L1 Cap O slash	Cadena	Ø	
ISO L1 Cap O tilde	Cadena	Õ	
ISO L1 Cap O umlaut	Cadena	Ö	
ISO L1 Cap THORN Icelandic	Cadena	Þ	
ISO L1 Cap U acute	Cadena	Ú	
ISO L1 Cap U circumflex	Cadena	Û	
ISO L1 Cap U grave	Cadena	Ù	
ISO L1 Cap U umlaut	Cadena	Ü	
ISO L1 Cap Y acute	Cadena	Ý	
ISO L1 Copyright	Cadena	©	
ISO L1 e acute	Cadena	é	
ISO L1 e circumflex	Cadena	ê	
ISO L1 e grave	Cadena	è	
ISO L1 e umlaut	Cadena	ë	
ISO L1 eth Icelandic	Cadena	ð	
ISO L1 Greater than	Cadena	>	
ISO L1 i acute	Cadena	í	
ISO L1 i circumflex	Cadena	î	
ISO L1 i grave	Cadena	ì	
ISO L1 i umlaut	Cadena	ï	
ISO L1 Less than	Cadena	<	
ISO L1 n tilde	Cadena	ñ	
ISO L1 o acute	Cadena	ó	
ISO L1 o circumflex	Cadena	ô	
ISO L1 o grave	Cadena	ò	
ISO L1 o slash	Cadena	ø	
ISO L1 o tilde	Cadena	õ	
ISO L1 o umlaut	Cadena	ö	
ISO L1 Quotation mark	Cadena	"	
ISO L1 Registered	Cadena	®	
ISO L1 sharp s German	Cadena	ß	
ISO L1 thorn Icelandic	Cadena	þ	
ISO L1 u acute	Cadena	ú	
ISO L1 u circumflex	Cadena	û	
ISO L1 u grave	Cadena	ù	
ISO L1 u umlaut	Cadena	ü	
ISO L1 y acute	Cadena	ý	

Constante	***Tipo***	***Valor***	***Comentario***
ISO L1 y umlaut	*Cadena*	*ÿ*	

Carpeta sistema

Constante	Tipo	Valor	Comentario
_o_Mac control panels	Entero largo	11	
_o_Mac extensions	Entero largo	10	
_o_Mac shutdown items_all	Entero largo	6	
_o_Mac shutdown items_user	Entero largo	7	
Applications or program files	Entero largo	16	
Desktop	Entero largo	15	
Documents folder	Entero largo	17	Carpeta "Documents" del usuario
Favorites Win	Entero largo	14	
Fonts	Entero largo	1	
Start menu Win_all	Entero largo	8	
Start menu Win_user	Entero largo	9	
Startup Win_all	Entero largo	4	
Startup Win_user	Entero largo	5	
System	Entero largo	0	
System Win	Entero largo	12	
System32 Win	Entero largo	13	
User preferences_all	Entero largo	2	
User preferences_user	Entero largo	3	

Códigos ASCII

Constante	Tipo	Valor	Comentario
ACK ASCII code	Entero largo	6	
At sign	Entero largo	64	
Backspace	Entero largo	8	
BEL ASCII code	Entero largo	7	
BS ASCII code	Entero largo	8	
CAN ASCII code	Entero largo	24	
Carriage return	Entero largo	13	
CR ASCII code	Entero largo	13	
DC1 ASCII code	Entero largo	17	
DC2 ASCII code	Entero largo	18	
DC3 ASCII code	Entero largo	19	
DC4 ASCII code	Entero largo	20	
DEL ASCII code	Entero largo	127	
DLE ASCII code	Entero largo	16	
Double quote	Entero largo	34	
EM ASCII code	Entero largo	25	
ENQ ASCII code	Entero largo	5	
Enter	Entero largo	3	
EOT ASCII code	Entero largo	4	
ESC ASCII code	Entero largo	27	
Escape	Entero largo	27	
ETB ASCII code	Entero largo	23	
ETX ASCII code	Entero largo	3	
FF ASCII code	Entero largo	12	
FS ASCII code	Entero largo	28	
GS ASCII code	Entero largo	29	
HT ASCII code	Entero largo	9	
LF ASCII code	Entero largo	10	
Line feed	Entero largo	10	
NAK ASCII code	Entero largo	21	
NBSP	Entero largo	202	
NUL ASCII code	Entero largo	0	
Period	Entero largo	46	
Quote	Entero largo	39	
RS ASCII code	Entero largo	30	
SI ASCII code	Entero largo	15	
SO ASCII code	Entero largo	14	
SOH ASCII code	Entero largo	1	
SP ASCII code	Entero largo	32	
Space	Entero largo	32	
STX ASCII code	Entero largo	2	
SUB ASCII code	Entero largo	26	
SYN ASCII code	Entero largo	22	
Tab	Entero largo	9	
US ASCII code	Entero largo	31	
VT ASCII code	Entero largo	11	

Colores

Constante	*Tipo*	*Valor*	*Comentario*
Black	*Entero largo*	*15*	
Blue	*Entero largo*	*6*	
Brown	*Entero largo*	*13*	
Dark blue	*Entero largo*	*5*	
Dark brown	*Entero largo*	*10*	
Dark green	*Entero largo*	*9*	
Dark grey	*Entero largo*	*11*	
Green	*Entero largo*	*8*	
Grey	*Entero largo*	*14*	
Light blue	*Entero largo*	*7*	
Light grey	*Entero largo*	*12*	
Orange	*Entero largo*	*2*	
Purple	*Entero largo*	*4*	
Red	*Entero largo*	*3*	
White	*Entero largo*	*0*	
Yellow	*Entero largo*	*1*	

Compresión de imagen

Constante	*Tipo*	*Valor*	*Comentario*

Comunicaciones

Constante	*Tipo*	*Valor*	*Comentario*
Data bits 5	*Entero largo*	*0*	
Data bits 6	*Entero largo*	*2048*	
Data bits 7	*Entero largo*	*1024*	
Data bits 8	*Entero largo*	*3072*	
MacOS printer port	*Entero largo*	*0*	
MacOS serial port	*Entero largo*	*1*	
Parity even	*Entero largo*	*12288*	
Parity none	*Entero largo*	*0*	
Parity odd	*Entero largo*	*4096*	
Protocol DTR	*Entero largo*	*30*	
Protocol none	*Entero largo*	*0*	
Protocol XONXOFF	*Entero largo*	*20*	
Speed 115200	*Entero largo*	*1022*	
Speed 1200	*Entero largo*	*94*	
Speed 1800	*Entero largo*	*62*	
Speed 19200	*Entero largo*	*4*	
Speed 230400	*Entero largo*	*1021*	
Speed 2400	*Entero largo*	*46*	
Speed 300	*Entero largo*	*380*	
Speed 3600	*Entero largo*	*30*	
Speed 4800	*Entero largo*	*22*	
Speed 57600	*Entero largo*	*0*	
Speed 600	*Entero largo*	*189*	
Speed 7200	*Entero largo*	*14*	
Speed 9600	*Entero largo*	*10*	
Stop bits one	*Entero largo*	*16384*	
Stop bits one and a half	*Entero largo*	*-32768*	
Stop bits two	*Entero largo*	*-16384*	

Crear ventana

Constante	Tipo	Valor	Comentario
_o_Compositing Mode	Entero largo	4096	*** Constante obsoleta ***
_o_Has toolbar button Mac	Entero largo	8192	*** Constante obsoleta ***
Alternate dialog box	Entero largo	3	Puede utilizarse en ventana flotante
Has full screen mode Mac	Entero largo	65536	
Has grow box	Entero largo	4	
Has highlight	Entero largo	1	
Has window title	Entero largo	2	
Has zoom box	Entero largo	8	
Modal dialog box	Entero largo	1	
Movable dialog box	Entero largo	5	Puede utilizarse en ventana flotante
Palette window	Entero largo	1984	Puede utilizarse en ventana flotante
Plain dialog box	Entero largo	2	Puede utilizarse en ventana flotante
Plain fixed size window	Entero largo	4	
Plain no zoom box window	Entero largo	0	
Plain window	Entero largo	8	
Pop up window	Entero largo	32	
Resizable sheet window	Entero largo	34	
Round corner window	Entero largo	16	
Sheet window	Entero largo	33	
Texture appearance	Entero largo	2048	

DEFINIR COLORES RVA

Constante	Tipo	Valor	Comentario
Background color	Entero largo	-2	
Background color none	Entero largo	-16	Esta constante puede ser utilizada únicamente con los parámetros colorFondo y colorFondoAlt.
Dark shadow color	Entero largo	-3	
Disable highlight item color	Entero largo	-11	
Foreground color	Entero largo	-1	
Highlight menu background color	Entero largo	-9	
Highlight menu text color	Entero largo	-10	
Highlight text background color	Entero largo	-7	
Highlight text color	Entero largo	-8	
Light shadow color	Entero largo	-4	

Destinos de búsqueda

Constante	Tipo	Valor	Comentario
Description in text format	Entero largo	0	
Description in XML format	Entero largo	1	
Into current selection	Entero largo	0	
Into named selection	Entero largo	2	
Into set	Entero largo	1	
Into variable	Entero largo	3	

Días y meses

Constante	Tipo	Valor	Comentario
April	Entero largo	4	
August	Entero largo	8	
December	Entero largo	12	
February	Entero largo	2	
Friday	Entero largo	6	
January	Entero largo	1	
July	Entero largo	7	
June	Entero largo	6	
March	Entero largo	3	
May	Entero largo	5	
Monday	Entero largo	2	
November	Entero largo	11	
October	Entero largo	10	
Saturday	Entero largo	7	
September	Entero largo	9	
Sunday	Entero largo	1	
Thursday	Entero largo	5	
Tuesday	Entero largo	3	
Wednesday	Entero largo	4	

Diccionarios

Nota de compatibilidad: *estas constantes corresponden a los números de los diccionarios "Cordial", que ya no son soportados por 4D a partir de la versión 14. Estas constantes se mantiene por razones de compatibilidad (un diccionario Hunspell equivalente se utiliza internamente).*

Constante	Tipo	Valor	Comentario

Documentos sistema

Constante	Tipo	Valor	Comentario
Absolute path	Entero largo	2	El array documentos contiene las rutas de acceso absolutas
Alias selection	Entero largo	8	Autoriza la selección de atajos (Windows) o de alias (Mac OS) como documentos. Por defecto, si no se utiliza esta constante no se utiliza, cuando se seleccione un alias o atajo, el comando devolverá la ruta de acceso del elemento objetivo. Cuando pase la constante, el comando devuelve la ruta del alias o del atajo.
Delete only if empty	Entero largo	0	Elimina la carpeta sólo cuando está vacía
Delete with contents	Entero largo	1	Elimina la carpeta junto con todo su contenido
Document unchanged	Entero largo	0	Ningún proceso
Document with CR	Entero largo	3	Las líneas de ruptura se convierten al formato OS X: CR (retorno de carro)
Document with CRLF	Entero largo	2	Las líneas de ruptura se convierten al formato Windows: CRLF (retorno de carro + salto de línea)
Document with LF	Entero largo	4	Las líneas de ruptura se convierten al formato Unix: LF (salto de línea)
Document with native format	Entero largo	1	(Por defecto) las líneas de ruptura se convierten al formato nativo del sistema operativo: CR (retorno de carro) en OS X, CRLF (retorno de carro + salto de línea) en Windows
File name entry	Entero largo	32	Autoriza al usuario a introducir un nombre de archivo en una caja de diálogo "Guardar como". No se guardan los archivos, el desarrollador debe crear un archivo en respuesta a esta acción (la variable sistema Documento se actualiza). En este contexto, el parámetro directorio puede contener la ruta a un archivo en lugar de a un directorio. El nombre del archivo se utilizará como nombre de archivo sugerido en el campo de texto Guardar como. El directorio padre se utilizará como ruta por defecto. Esta constante tiene un valor diferente en función del sistema operativo desde el cual se llama. Puede utilizarse para crear rutas válidas sin tener en cuenta la plataforma de ejecución.
Folder separator	Cadena	Windows="\" Mac OS=":"	**Nota:** esta constante puede utilizarse únicamente en la aplicaciones 4D ejecutadas en modo Unicode (no es soportada en modo de compatibilidad no Unicode).
Get pathname	Entero largo	3	
Ignore invisible	Entero largo	8	Los documentos invisibles no se lista
Is a document	Entero largo	1	
Is a folder	Entero largo	0	
Multiple files	Entero largo	1	Autoriza la selección simultánea de varios archivos utilizando las combinaciones **Mayús+clic** (selección adyacente) y **Ctrl+clic** (Windows) o **Comando+clic** (Mac OS). En este caso, el parámetro seleccionado, si se pasa, contiene la lista de todos los archivos seleccionados. Por defecto, si esta constante no se utiliza, el comando no permitirá la selección de archivos múltiples.
Package open	Entero largo	2	(Mac OS únicamente): autoriza la apertura de paquetes como carpetas y por lo tanto la visualización/selección de sus contenidos. Por defecto, si no se utiliza esta constante, el comando no permitirá la apertura de paquetes.
Package selection	Entero largo	4	(Mac OS únicamente): autoriza la selección de paquetes como entidades. Por defecto, si no se utiliza esta constante, el comando no permitirá la selección de paquetes como tales.
Path is POSIX	Entero largo	1	La ruta se expresa utilizando la sintaxis de Posix
Path is system	Entero largo	0	(Por defecto) La ruta se expresa utilizando la sintaxis actual del sistema (Windows o macOS)
Posix path	Entero largo	4	El array documentos contiene las rutas de acceso al formato POSIX
Read and write	Entero largo	0	Modo por defecto
Read mode	Entero largo	2	
Recursive parsing	Entero largo	1	El array documentos contiene los archivos y todas las subcarpetas de la carpeta especificada
Use sheet window	Entero largo	16	(Mac OS únicamente): muestra la caja de diálogo de selección en forma de una ventana hoja (esta opción se ignora en Windows). Las ventanas hojas son específicas para la interfaz Mac OS X con animación gráfica (para mayor información, consulte la sección **Tipos de ventanas (Compatibilidad)**). Por defecto, si esta constante no se utiliza, el comando mostrará una caja de diálogo estándar.
Write mode	Entero largo	1	

Entorno 4D

Constante	Tipo	Valor	Comentario
_o_4D Interpreted desktop	Entero largo	2	
_o_Extras folder	Entero largo	2	Carpeta con contenido personalizado descargado para cada equipo cliente. **Nota de compatibilidad:** a partir de la versión 11.2 de 4D v11 SQL, no es recomendable utilizar las carpeta **Extras** para la comunicación personalizada entre el servidor y los equipos distantes. Ahora se recomienda para este propósito utilizar la carpeta **Resources** (ver la descripción de la carpeta **Resources** actual a continuación. Sin embargo, la carpeta **Extras** aún es soportada por 4D Server para mantener la compatibilidad de las aplicaciones existentes. **Nota:** si la carpeta **Extras** no existe en la base, se creará ejecutando el comando **Get 4D folder** con la constante Extras Folder.
_o_Full Version	Entero largo	0	
4D Client database folder	Entero largo	3	
4D Desktop	Entero largo	3	
4D Local mode	Entero largo	0	
4D Remote mode	Entero largo	4	
4D Server	Entero largo	5	
4D Volume desktop	Entero largo	1	
64 bit version	Entero largo	1	
Active 4D Folder	Entero largo	0	
Backup configuration file	Entero largo	1	Archivo Backup.xml, almacenado en la carpeta Preferencias/Backup junto al archivo de estructura de la base
Backup log file	Entero largo	13	Archivo de historial de la copia de seguridad actual. Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Build application log file	Entero largo	14	Archivo de historial actual en formato xml del generador de aplicaciones. Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Compacting log file	Entero largo	6	Archivo de historial de la compactación más reciente de la base, creada por el comando **Compact data file** o por el Centro de seguridad y mantenimiento (CSM). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Current localization	Entero largo	1	Lenguaje actual de la aplicación: lenguaje por defecto o lenguaje definido vía el comando **SET DATABASE LOCALIZATION**.
Current resources folder	Entero largo	6	
Data folder	Entero largo	9	
Database folder	Entero largo	4	
Database folder Unix syntax	Entero largo	5	
Debug log file	Entero largo	12	Archivo de historial creado por el comando **SET DATABASE PARAMETER**(Debug log recording). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Default localization	Entero largo	0	Lenguaje definido automáticamente por 4D al inicio en función de la carpeta Resources y del entorno sistema (no modificable)
Demo version	Entero largo	0	
Diagnostic log file	Entero largo	11	Archivo de historial creado por **SET DATABASE PARAMETER**(Diagnostic log recording). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Full method text	Entero largo	1	
Highlighted method text	Entero largo	2	
HTML Root folder	Entero largo	8	

Constante	Tipo	Valor	Comentario
HTTP debug log file	Entero largo	9	Archivo de historial creado por el comando **WEB SET OPTION**(Web debug log). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Internal 4D localization	Entero largo	3	Lenguaje utilizado por 4D para ordenaciones y comparaciones de textos (definido en las Preferencias de la aplicación).
Last backup file	Entero largo	2	Último archivo de copia de seguridad, llamado <nombreBase>[bkpNum].4BK, almacenado en una ubicación personalizada
Licenses folder	Entero largo	1	
Logs folder	Entero largo	7	
Merged application	Entero largo	2	La version es una aplicación fusionada con 4D Volume Desktop
Processes only	Entero largo	1	Devuelve sólo la lista de procesos
Repair log file	Entero largo	7	Archivo de historial de las reparaciones realizadas a la base por el Centro de mantenimiento y seguridad (CMS). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.
Request log file	Entero largo	10	Archivo de peticiones cliente/servidor estándar (excluyendo peticiones web) creado por los comandos **SET DATABASE PARAMETER**(4D Server log recording) o **SET DATABASE PARAMETER**(Client log recording). Si se ejecuta en el servidor, se devuelve el historial del servidor. Almacenado en la carpeta Logs en el servidor. Si se ejecuta en el cliente, se devuelve el historial del cliente. Almacenado en la carpeta Logs en el cliente. Si no existe ningún archivo de historial, se devuelve una ruta vacía.
Sessions only	Entero largo	2	Devuelve sólo la lista de sesión de usuario
Structure settings	Entero largo	0	Acceso a las "propiedades estructura" (valor por defecto si el parámetro se omite). En este modo, los valores de selector utilizables son idénticos a los del modo estándar.
User settings	Entero largo	1	Acceso a las "propiedades usuario". En este modo, sólo ciertas llaves son utilizables en el parámetro selector.
User settings file	Entero largo	3	El archivo settings.4DSettings para los archivos de datos, almacenado en la carpeta Preferencias junto al archivo estructura de la base si se activa
User settings file for data	Entero largo	4	settings.4DSettings para el archivo de datos actual, almacenado en la carpeta Preferencias junto al archivo de datos.
User settings for data	Entero largo	2	Acceso a "Configuración usuario para archivo de datos, que es, configuración usuario almacenada en el mismo nivel que el archivo de datos. En este modo, sólo ciertas llaves se pueden utilizar con el parámetro selector (mismo subconjunto que la configuración usuario)
User system localization	Entero largo	2	Lenguaje definido por el usuario actual del sistema.
Verification log file	Entero largo	5	Archivos de historial creados por los comandos **VERIFY CURRENT DATA FILE** y **VERIFY DATA FILE** o el Centro de mantenimiento y seguridad (CMS). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha realizado ninguna verificación o no existe ningún archivo de historial, se devuelve una ruta vacía. No se generan errores.
Web request log file	Entero largo	8	Archivo de historial creado por el comando **WEB SET OPTION**(Web log recording). Almacenado en la carpeta Logs junto al archivo de estructura de la base. Si no se ha creado ningún archivo de historial o no existe, se devuelve una ruta vacía. No se generan errores.

Estado del proceso

Constante	*Tipo*	*Valor*	*Comentario*
Aborted	*Entero largo*	*-1*	
Delayed	*Entero largo*	*1*	
Does not exist	*Entero largo*	*-100*	
Executing	*Entero largo*	*0*	
Hidden modal dialog	*Entero largo*	*6*	
Paused	*Entero largo*	*5*	
Waiting for input output	*Entero largo*	*3*	
Waiting for internal flag	*Entero largo*	*4*	
Waiting for user event	*Entero largo*	*2*	

Estilos de fuente

Las constantes con prefijo _O_ son obsoletas y ya no deben utilizarse.

Constante	Tipo	Valor	Comentario
_o_Condensed	*Entero largo*	*32*	
_o_Extended	*Entero largo*	*64*	
_o_Outline	*Entero largo*	*8*	
_o_Shadow	*Entero largo*	*16*	
Automatic style sheet	*Cadena*	*__automatic__*	*Se utiliza de forma predeterminada para todos los objetos*
Automatic style sheet_additional	*Cadena*	*__automatic_additional_text__*	*Soportado por los textos estáticos, campos y variables solamente. Se utiliza para texto adicional en las cajas de diálogo.*
Automatic style sheet_main	*Cadena*	*__automatic_main_text__*	*Soportado por los textos estáticos, campos y variables solamente. Se utiliza para texto principal en las cajas de diálogo.*
Bold	*Entero largo*	*1*	
Bold and Italic	*Entero largo*	*3*	
Bold and Underline	*Entero largo*	*5*	
Italic	*Entero largo*	*2*	
Italic and Underline	*Entero largo*	*6*	
Plain	*Entero largo*	*0*	
Underline	*Entero largo*	*4*	

Euro monedas

Constante	*Tipo*	*Valor*	*Comentario*
Austrian Schilling	*Cadena*	*ATS*	
Belgian Franc	*Cadena*	*BEF*	
Deutsche Mark	*Cadena*	*DEM*	
Euro	*Cadena*	*EUR*	
Finnish Markka	*Cadena*	*FIM*	
French Franc	*Cadena*	*FRF*	
Greek Drachma	*Cadena*	*GRD*	
Irish Pound	*Cadena*	*IEP*	
Italian Lira	*Cadena*	*ITL*	
Luxembourg Franc	*Cadena*	*LUF*	
Netherlands Guilder	*Cadena*	*NLG*	
Portuguese Escudo	*Cadena*	*PTE*	
Spanish Peseta	*Cadena*	*ESP*	

Eventos (Modificadores)

Constante	Tipo	Valor	Comentario
Activate window bit	*Entero largo*	*0*	
Activate window mask	*Entero largo*	*1*	
Caps lock key bit	*Entero largo*	*10*	*Windows y OS X*
Caps lock key mask	*Entero largo*	*1024*	*Windows y OS X*
Command key bit	*Entero largo*	*8*	*Tecla Ctrl en Windows, tecla Comando en OS X)*
Command key mask	*Entero largo*	*256*	*Tecla Ctrl en Windows, Tecla Comando en OS X*
Control key bit	*Entero largo*	*12*	*Tecla Ctrl en OS X, o clic derecho en Windows y OS X*
Control key mask	*Entero largo*	*4096*	*Tecla Ctrl bajo OS X, o clic derecho en Windows y OS X*
Mouse button bit	*Entero largo*	*7*	
Mouse button mask	*Entero largo*	*128*	
Option key bit	*Entero largo*	*11*	*Tecla Alt (también llamada Opción en OS X)*
Option key mask	*Entero largo*	*2048*	*Tecla Alt (también llamada Opción en OS X)*
Right control key bit	*Entero largo*	*15*	
Right control key mask	*Entero largo*	*32768*	
Right option key bit	*Entero largo*	*14*	
Right option key mask	*Entero largo*	*16384*	
Right shift key bit	*Entero largo*	*13*	
Right shift key mask	*Entero largo*	*8192*	
Shift key bit	*Entero largo*	*9*	*Windows y OS X*
Shift key mask	*Entero largo*	*512*	*Windows y OS X*

Eventos (tipos)

Constante	*Tipo*	*Valor*	*Comentario*
Activate event	*Entero largo*	*8*	
Auto key event	*Entero largo*	*5*	
Disk event	*Entero largo*	*7*	
Key down event	*Entero largo*	*3*	
Key up event	*Entero largo*	*4*	
Mouse down event	*Entero largo*	*1*	
Mouse up event	*Entero largo*	*2*	
Null event	*Entero largo*	*0*	
Operating system event	*Entero largo*	*15*	
Update event	*Entero largo*	*6*	

Eventos de la base

Constante	Tipo	Valor	Comentario
On after host database exit	Entero largo	4	El **Semaphore** de la base local acaba de terminar su ejecución
On after host database startup	Entero largo	2	El **Método base On Startup** de la base local acaba de terminar su ejecución
On application background move	Entero largo	1	La aplicación 4D pasa al fondo
On application foreground move	Entero largo	2	La aplicación 4D pasa al primer plano
On before host database exit	Entero largo	3	La base local se está cerrando. El **Semaphore** de la base local aún no se ha llamado. El **Semaphore** de la base loal no se llama mientras el **Método base On Host Database Event** del componente se esté ejecutando
On before host database startup	Entero largo	1	La base local se ha iniciado. El **Método base On Startup** de la base local aún no se ha llamado. El método base On Startup de la base local no se llama mientras el **Método base On Host Database Event** del componente se esté ejecutando

Eventos trigger

Constante	*Tipo*	*Valor*	*Comentario*
_o_On Loading Record Event	*Entero largo*	*4*	
On Deleting Record Event	*Entero largo*	*3*	
On Saving Existing Record Event	*Entero largo*	*2*	
On Saving New Record Event	*Entero largo*	*1*	

Expresiones

Constante	Tipo	Valor	Comentario
MAXINT	*Entero largo*	*32767*	
MAXLONG	*Entero largo*	*2147483647*	
MAXTEXTLENBEFOREV11	*Entero largo*	*32000*	

Firmas del sistema estándar

Las firmas estándar del sistema son tipos de archivos estándar de 4 caracteres , tipos de recursos, tipos estándar de información almancenados en el portapapeles.

Constante	Tipo	Valor	Comentario
Picture document	*Cadena*	*PICT*	
Text document	*Cadena*	*TEXT*	
Windows MIDI document	*Cadena*	*MID*	
Windows sound document	*Cadena*	*WAV*	
Windows video document	*Cadena*	*AVI*	

Form event

Constante	Tipo	Valor	Comentario
_o_On Mac toolbar button	Entero largo	55	*** Constante obsoleta ***
On Activate	Entero largo	11	La ventana del formulario se convierte en la ventana del primer plano
On After Edit	Entero largo	45	El contenido del objeto editable que tiene el foco acaba de ser modificado
On After Keystroke	Entero largo	28	Un carácter está apunto de introducirse en el objeto que tiene el foco. **Get edited text** devuelve el contenido incluyendo este carácter
On After Sort	Entero largo	30	(List box únicamente) Se acaba de efectuar una ordenación estándar en una columna del list box
On Alternative Click	Entero largo	38	• Botones 3D: el área "flecha" de un botón 3D recibe un clic • List boxes: en una columna de un array objeto, un botón de puntos suspensivos (atributo "alternateButton") recibe un clic **Nota**: los botones de elipses están disponibles sólo para versiones v15 o superiores.
On Before Data Entry	Entero largo	41	(List box únicamente) Una celda de list box está a punto de pasar a modo edición
On Before Keystroke	Entero largo	17	Un carácter está a punto de introducirse en el objeto que tiene el foco. **Get edited text** devuelve el texto del objeto sin este carácter.
On Begin Drag Over	Entero largo	46	Se va a arrastrar un objeto
On Begin URL Loading	Entero largo	47	(Áreas web únicamente) Un nuevo URL se carga en el área web
On bound variable change	Entero largo	54	Se modifica la variable relacionada a un subformulario.
On Clicked	Entero largo	4	Ocurre un clic sobre un objeto
On Close Box	Entero largo	22	Se ha hecho clic en la casilla de cerrar la ventana
On Close Detail	Entero largo	26	Se cierra el formulario de entrada y regresa al formulario de salida
On Collapse	Entero largo	44	(Listas jerárquicas únicamente) Un elemento de la lista jerárquica se ha contraído vía un clic o una tecla
On Column Moved	Entero largo	32	(List box únicamente) El usuario mueve una columna de list box vía arrastrar y soltar
On Column Resize	Entero largo	33	(List box únicamente) El ancho de una columna de list box es modificado por un usuario con el ratón
On Data Change	Entero largo	20	Se han modificado los datos de un objeto
On Deactivate	Entero largo	12	La ventana del formulario deja de ser la ventana del primer plano
On Delete Action	Entero largo	58	(Listas jerárquicas y list box únicamente). El usuario solicita borrar un elemento.
On Display Detail	Entero largo	8	Un registro se va a mostrar en una lista o una línea se va a mostrar en un list box.
On Double Clicked	Entero largo	13	Un objeto ha recibido un doble clic
On Drag Over	Entero largo	21	Los datos pueden soltarse en un objeto
On Drop	Entero largo	16	Se han soltado datos en un objeto
On End URL Loading	Entero largo	49	(Áreas web únicamente) Se han cargado todos los recursos del URL
On Expand	Entero largo	43	(Listas jerárquicas únicamente) Se ha expandido un elemento de la lista jerárquica utilizando un clic o una tecla
On Footer Click	Entero largo	57	(List box únicamente) Un clic en el pie de un list box o de una columna de list box
On Getting Focus	Entero largo	15	Un objeto de formulario toma el foco
On Header	Entero largo	5	El encabezado del formulario se va a imprimir o a mostrar
On Header Click	Entero largo	42	(List box únicamente) Ocurre un clic en un encabezado de columna del list box
On Load	Entero largo	1	El formulario se muestra o se imprime
On Load Record	Entero largo	40	En modo entrada en lista, se carga un registro durante modificación (el usuario hace clic en una línea del registro y un campo pasa a modo edición)
On Long Click	Entero largo	39	(Botones 3D únicamente) Se hace clic en un botón 3D y el botón del ratón permanece presionado por un cierto tiempo

Constante	Tipo	Valor	Comentario
On Losing Focus	Entero largo	14	Un objeto de formulario está perdiendo el foco
On Menu Selected	Entero largo	18	Se ha seleccionado un comando de menú
On Mouse Enter	Entero largo	35	El cursor del ratón entra al área gráfica de un objeto
On Mouse Leave	Entero largo	36	El cursor del ratón sale del área gráfica de un objeto
On Mouse Move	Entero largo	37	El cursor del ratón se mueve al menos un píxel O cuando se presiona una tecla de modificación (Ctrl, Alt, Bloq mayús). Si el evento está seleccionado para un objeto únicamente, se genera sólo cuando el cursor se encuentra dentro del área gráfica del objeto
On Mouse Up	Entero largo	2	(Sólo imágenes) El usuario acaba de liberar el botón izquierdo del ratón en un objeto Imagen
On Open Detail	Entero largo	25	El formulario detallado asociado con el formulario de salida o con el listbox está apunto de ser abierto
On Open External Link	Entero largo	52	(Áreas web únicamente) Se ha abierto un URL externo en el navegador
On Outside Call	Entero largo	10	El formulario recibe una llamada **POST OUTSIDE CALL**
On Page Change	Entero largo	56	Al cambiar de página actual en el formulario
On Plug in Area	Entero largo	19	Un objeto externo solicitó que se ejecute su método de objeto
On Printing Break	Entero largo	6	Se va a imprimir una de las áreas de ruptura del formulario
On Printing Detail	Entero largo	23	Se va a imprimir el área de detalle del formulario
On Printing Footer	Entero largo	7	Se va a imprimir el área de pie de página del formulario
On Resize	Entero largo	29	La ventana del formulario se redimensiona
On Row Moved	Entero largo	34	(List box únicamente) El usuario mueve una fila de un list box vía arrastrar y soltar
On Scroll	Entero largo	59	El usuario desplaza el contenido de un campo o de una variable imagen utilzando el ratón o una tecla.
On Selection Change	Entero largo	31	• List box: se modifica la selección actual de líneas o columnas • Registros en lista: se modifica el registro actual o la selección actual de líneas en un formulario listado o en un subformulario • Lista jerárquica: la selección en la lista se modifica luego de un clic o de presionar una tecla • Variable o campo editable: la selección de texto o la posición del cursor en el área se modifica al hacer clic o presionar una tecla.
On Timer	Entero largo	27	El número de tics definido por el comando **SET TIMER** se ha pasado
On Unload	Entero largo	24	El formulario se cierra y libera
On URL Filtering	Entero largo	51	(Áreas web únicamente) Un URL fue bloqueado por el área web
On URL Loading Error	Entero largo	50	(Áreas web únicamente) Ocurrió un error cuando se cargaba el URL
On URL Resource Loading	Entero largo	48	(Áreas web únicamente) Se carga un nuevo recurso en el área web
On Validate	Entero largo	3	Se ha valido la entrada de datos en el registro
On VP Ready	Entero largo	9	(Áreas 4D View Pro únicamente) La carga del área 4D View Pro está completa
On Window Opening Denied	Entero largo	53	(Áreas web únicamente) Se ha bloqueado una ventana pop-up

Formato de sistema

Constante	*Tipo*	*Valor*	*Comentario*
Currency symbol	*Entero largo*	*2*	*Símbolo de moneda (ej.: "$")*
Date separator	*Entero largo*	*13*	*Separador utilizado en formatos de fecha (ej.: "/")*
Decimal separator	*Entero largo*	*0*	*Separador decimal (ej.: ".")*
Short date day position	*Entero largo*	*15*	*Posición de día en el formato de fecha corto: "1" = izquierda, "2" = en el medio, "3" = a la derecha*
Short date month position	*Entero largo*	*16*	*Posición del mes en formato de fecha corto: "1" = izquierda, "2" = en el medio, "3" = a la derecha*
Short date year position	*Entero largo*	*17*	*Posición del año en el formato de fecha corto: "1" = izquierda, "2" = medio, "3" = derecha*
System date long pattern	*Entero largo*	*8*	*Formato de salida de fecha largo "dddd MMMM yyyy"*
System date medium pattern	*Entero largo*	*7*	*Formato de salida de fecha medio en la forma "dddd MMMM yyyy"*
System date short pattern	*Entero largo*	*6*	*Formato de salida de fecha corto en la forma "dddd MMMM yyyy"*
System time AM label	*Entero largo*	*18*	*Etiqueta adicional para una hora antes del medio día en formatos de 12 horas (ej.: "Mañana")*
System time long pattern	*Entero largo*	*5*	*Formato de salida de hora largo en la forma "HH:MM:SS"*
System time medium pattern	*Entero largo*	*4*	*Formato de salida de hora medio en la forma "HH:MM:SS"*
System time PM label	*Entero largo*	*19*	*Etiqueta adicional para una hora luego del medio día en formatos de 12 horas (ej.: "tarde")*
System time short pattern	*Entero largo*	*3*	*Formato de salida de hora corto en forma "HH:MM:SS"*
Thousand separator	*Entero largo*	*1*	*Separador de miles (ej.: ",")*
Time separator	*Entero largo*	*14*	*Separador utilizado en formatos de hora (ej.: ":")*

Formatos de salida de fechas

Constante	*Tipo*	*Valor*	*Comentario*
Blank if null date	*Entero largo*	*100*	*"" en lugar de 0 en caso de valor nulo. Esta constante debe adicionarse al formato de visualización.*
Date RFC 1123	*Entero largo*	*10*	*Fri, 10 Sep 2010 13:07:20 GMT*
Internal date abbreviated	*Entero largo*	*6*	*29 dic, 2006*
Internal date long	*Entero largo*	*5*	*29 diciembre 2006*
Internal date short	*Entero largo*	*7*	*12/29/2006*
Internal date short special	*Entero largo*	*4*	*12/29/06 (pero 12/29/1896 o 12/29/2096)*
ISO Date	*Entero largo*	*8*	*2006-12-29T00:00:00 (obsoleto)*
ISO Date GMT	*Entero largo*	*9*	*2010-09-13T16:11:53Z*
System date abbreviated	*Entero largo*	*2*	*dom. 29 de 2006*
System date long	*Entero largo*	*3*	*domingo 29 diciembre 2006*
System date short	*Entero largo*	*1*	*12/29/2006*

Formatos de salida de hora

Constante	*Tipo*	*Valor*	*Comentario*
Blank if null time	*Entero largo*	*100*	*"" en lugar de 0*
HH MM	*Entero largo*	*2*	*01:02*
HH MM AM PM	*Entero largo*	*5*	*1:02 AM*
HH MM SS	*Entero largo*	*1*	*01:02:03*
Hour min	*Entero largo*	*4*	*1 hora 2 minutos*
Hour min sec	*Entero largo*	*3*	*1 hora 2 minutos 3 segundos*
ISO time	*Entero largo*	*8*	*0000-00-00T01:02:03*
Min sec	*Entero largo*	*7*	*62 minutos 3 segundos*
MM SS	*Entero largo*	*6*	*62:03*
System time long	*Entero largo*	*11*	*1:02:03 AM HNEC (Mac únicamente)*
System time long abbreviated	*Entero largo*	*10*	*1•02•03 AM (Mac únicamente)*
System time short	*Entero largo*	*9*	*01:02:03*

Formatos de salida de imágenes

Constante	Tipo	Valor	Comentario
On background	*Entero largo*	*3*	
Replicated	*Entero largo*	*7*	
Scaled to fit	*Entero largo*	*2*	
Scaled to fit prop centered	*Entero largo*	*6*	
Scaled to fit proportional	*Entero largo*	*5*	
Truncated centered	*Entero largo*	*1*	
Truncated non centered	*Entero largo*	*4*	

Fórmulas

Constante	*Tipo*	*Valor*	*Comentario*
Formula in with virtual structure	*Cadena*	*1*	*La fórmula contiene nombres personalizados (virtual). Por defecto, la fórmula devuelta contiene nombres reales.*
Formula out with tokens	*Cadena*	*4*	*La fórmula devuelta debe contener texto tokenizado (por ejemplo: Cxx).*
Formula out with virtual structure	*Cadena*	*2*	*La fórmula devuelta debe contener nombres personalizados (virtual).*

Funciones matemáticas

Constante	*Tipo*	*Valor*	*Comentario*
Degree	*Real*	*0.0174532925199432958*	
e number	*Real*	*2.71828182845904524*	
Pi	*Real*	*3.141592653589793239*	
Radian	*Real*	*57.29577951308232088*	

Gestión

Constante	Tipo	Valor	Comentario
Cache priority high	*Entero largo*	*1000*	
Cache priority low	*Entero largo*	*-900*	
Cache priority normal	*Entero largo*	*0*	*Define la prioridad de la caché a su valor por defecto*
Cache priority very high	*Entero largo*	*30000*	
Cache priority very low	*Entero largo*	*-1*	

Historial de eventos

Constante	Tipo	Valor	Comentario
Error message	Entero largo	2	
Information message	Entero largo	0	
Into 4D commands log	Entero largo	3	Indica a 4D grabar el mensaje en el archivo de historial de los comandos de 4D, si este archivo se ha activado. El archivo de historial de comandos de 4D puede activarse utilizando el comando **SET DATABASE PARAMETER** (selector 34). **Nota:** los archivos de historial de 4D se agrupan en la carpeta **Logs**, creada junto al archivo de estructura de la base (ver el comando **Get 4D folder**).
Into 4D debug message	Entero largo	1	Indica a 4D enviar el mensaje al entorno de depuración del sistema. El resultado depende de la plataforma: • Bajo Mac OS: el comando envía el mensaje a la Consola • Bajo Windows: el comando envía el mensaje como un mensaje de depuración. Para poder leer este mensaje, debe tener Microsoft Visual Studio o DebugView para Windows (http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx)
Into 4D diagnostic log	Entero largo	5	Le indica a 4D poner el mensaje en el archivo de diagnóstico de 4D, si este archivo está activo. El archivo de diagnóstico puede activarse con ayuda del comando **SET DABABASE PARAMETER** (selector 79).
Into 4D request log	Entero largo	2	Indica a 4D grabar el mensaje en el archivo de historial de peticiones de 4D, si este archivo ha sido activado
Into Windows log events	Entero largo	0	Indica a 4D enviar el mensaje "Log events" de Windows. Este historial recibe y almacena los mensajes que vienen de las aplicaciones en ejecución. En este caso, puede definir el nivel de importancia del mensaje vía el parámetro opcional importancia (ver a continuación). **Notas:** • Para que esta funcionalidad esté disponible, el servicio Log Events de Windows debe estar en ejecución. • Bajo Mac OS, el comando no hace nada con este tipo de salida
Warning message	Entero largo	1	

HTTP Client

Constante	Tipo	Valor	Comentario
HTTP basic	Entero largo	1	Utilizar el método de autenticación BASIC
HTTP compression	Entero largo	6	valor = 0 (no comprimir) ó 1 (comprimir). Por defecto: 0 Esta opción activa o desactiva el mecanismo de compresión de las peticiones entre el cliente y el servidor, para acelerar los intercambios. Cuando este mecanismo está activo, el cliente HTTP utiliza la compresión deflate o GZIP en función de la respuesta del servidor.
HTTP DELETE method	Cadena	DELETE	Ver el RFC 2616
HTTP digest	Entero largo	2	Utilizar el método de autenticación DIGEST
HTTP display auth dial	Entero largo	4	valor = 0 (no mostrar el diálogo) o 1 (mostrar el diálogo). Por defecto: 0 Esta opción controla la visualización de la caja de diálogo de autenticación al ejecutar el comando **HTTP Get** o **HTTP Request**. Por defecto, este comando no provoca la visualización de la caja de diálogo, normalmente debe utilizar el comando **HTTP AUTHENTICATE**. Sin embargo, si desea que aparezca una caja de diálogo de autenticación del usuario para que introduzca su nombre de usuario y contraseña, pase 1 en valor. La caja de diálogo aparece sólo si la solicitud requiere autenticación.
HTTP follow redirect	Entero largo	2	valor = 0 (no acepta redirecciones) o 1 (acepta redirecciones). Valor por defecto = 2
HTTP GET method	Cadena	GET	Ver el RFC 2616. Equivale a utilizar el comando **HTTP Get**
HTTP HEAD method	Cadena	HEAD	Ver el RFC 2616
HTTP max redirect	Entero largo	3	valor = número máximo de redirecciones aceptadas Valor por defecto = 2
HTTP OPTIONS method	Cadena	OPTIONS	Ver el RFC 2616
HTTP POST method	Cadena	POST	Ver el RFC 2616
HTTP PUT method	Cadena	PUT	Ver el RFC 2616
HTTP reset auth settings	Entero largo	5	valor = 0 (no borrar la información) ó 1 (borrar). Por defecto: 0 Esta opción permite indicar a 4D memorizar la información de autenticación del usuario (nombre de usuario, contraseña, método, etc.) Con el fin de volver a usarlos más adelante. Por defecto, esta información se borra después de cada ejecución del comando **HTTP Get** o **HTTP Request**. Pase 0 en valor para memorizarlos y para borrarlos. Tenga en cuenta que cuando se pasa 0, la información se conserva durante la sesión, pero no se guarda.
HTTP timeout	Entero largo	1	valor = timeout de la petición del cliente, expresada en segundos. El time out es el tiempo de espera del cliente HTTP en caso de no respuesta por parte del servidor. Al final de este período, el cliente cierra la sesión, la petición se pierde. Por defecto, este tiempo es de 120 segundos. Puede cambiarse en función de características específicas (estado de la red, características de la aplicación, etc).
HTTP TRACE method	Cadena	TRACE	Ver el RFC 2616

Interfaz de la plataforma

Constante	*Tipo*	*Valor*	*Comentario*

LDAP

Constante	*Tipo*	*Valor*	*Comentario*
LDAP all levels	*Cadena*	*sub*	*Buscar en el elemento raíz definido por dnRootEntry y en todas las ramas siguientes*
LDAP password MD5	*Entero largo*	*0*	*(Por defecto) Enviar contraseña encriptada en MD5*
LDAP password plain text	*Entero largo*	*1*	*Envío de contraseña sin encripción (conexión TLS recomendada)*
LDAP root and next	*Cadena*	*one*	*Buscar en el nivel de entrada raíz definido por dnRootEntry y en las entradas directamente posteriores en un nivel*
LDAP root only	*Cadena*	*base*	*Buscar únicamente en el elemento raíz definido por dnRootEntry (se omite por defecto)*

Licencia disponible

Constante	*Tipo*	*Valor*	*Comentario*
4D Client SOAP license	*Entero largo*	*808465465*	
4D Client Web license	*Entero largo*	*808465209*	
4D for OCI license	*Entero largo*	*808465208*	
4D Mobile license	*Entero largo*	*808464439*	
4D Mobile Test license	*Entero largo*	*808465719*	
4D ODBC Pro license	*Entero largo*	*808464946*	
4D SOAP license	*Entero largo*	*808465464*	
4D SOAP local license	*Entero largo*	*808531000*	
4D SOAP one connection license	*Entero largo*	*825242680*	
4D SQL Server license	*Entero largo*	*808464949*	
4D SQL Server local license	*Entero largo*	*808530485*	
4D SQL Server one conn license	*Entero largo*	*825242165*	
4D View license	*Entero largo*	*808465207*	
4D Web license	*Entero largo*	*808464945*	
4D Web local license	*Entero largo*	*808530481*	
4D Web one connection license	*Entero largo*	*825242161*	
4D Write license	*Entero largo*	*808464697*	

List box cálculo pie

Constante	Tipo	Valor	Comentario
Listbox footer std deviation	Entero largo	7	Utilizable con las columnas de tipo numérico u hora (list boxes de tipo array únicamente) Tipo de resultado por defecto: Real
lk footer average	Entero largo	6	Utilizable con las columnas de tipo numérico, hora Tipo de resultado por defecto:Real
lk footer count	Entero largo	5	Utilizable con las columnas de tipo numérico, texto, fecha, hora, booleano, imagen Tipo de resultado por defecto: Entero largo
lk footer custom	Entero largo	1	Ningún cálculo es efectuado por 4D. La variable del pie debe calcularse por programación. Típo por defecto del resultado del cálculo: tipo de la variable
lk footer max	Entero largo	3	Utilizable con las columnas de tipo numérico, fecha, hora, booleano Tipo de resultado por defecto: tipo del array o campo de la columna
lk footer min	Entero largo	2	Utilizable con las columnas de tipo numérico, fecha, hora, booleano Tipo por defecto del resultado: tipo del array o campo de la columna
lk footer sum	Entero largo	4	Utilizable con las columnas de tipo numérico, hora, booleano Tipo de resultado por defecto: tipo del array o campo de la columna
lk footer sum squares	Entero largo	9	Utilizable con las columnas de tipo numérico, hora (listbox de tipo array únicamente) Tipo por defecto del resultado: Real
lk footer variance	Entero largo	8	Utilizable con las columnas de tipo numérico, hora (listbox de tipo array únicamente) Tipo por defecto del resultado: Real

Listas jerárquicas

Constante	Tipo	Valor	Comentario
_o_Macintosh node	Entero largo	860	
_o_Use PICT resource	Entero largo	65536	
_o_Windows node	Entero largo	138	
Additional text	Cadena	4D_additional_text	Esta constante se utiliza para agregar texto a la derecha del elemento refElem. Este título adicional siempre se mostrará en la parte derecha de la lista, incluso cuando el usuario mueva el cursor de desplazamiento horizontal. Cuando utilice esta constante, pase el texto que se mostrará en valor.
Associated standard action	Cadena	4D_standard_action_name	Asociar una acción estándar con el refElem. En este caso, debe pasar en el parámetro valor un nombre de acción estándar con un parámetro, por ejemplo "fontSize?value=10pt". Para más información, consulte la sección **Acciones estándar** del manual de Diseño.
Use PicRef	Entero largo	131072	

Listbox

Constante	Tipo	Valor	Comentario
_o_lk display hor scrollbar	Entero largo	2	***Constante obsoleta*** Utilizar el comando **OBJECT GET SCROLLBAR**.
_o_lk display ver scrollbar	Entero largo	4	***Constante obsoleta*** Utilizar el comando **OBJECT GET SCROLLBAR**.
_o_lk footer height	Entero largo	9	***Constante obsoleta*** Utilizar el comando **LISTBOX Get footers height**.
_o_lk header height	Entero largo	1	***Constante obsoleta*** Utilizar el comando **LISTBOX Get headers height**.
_o_lk hor scrollbar position	Entero largo	6	***Constante obsoleta*** Utilizar el comando **OBJECT GET SCROLL POSITION**.
_o_lk ver scrollbar position	Entero largo	7	***Constante obsoleta*** Utilizar el comando **OBJECT GET SCROLL POSITION**.
lk add to selection	Entero largo	1	La línea seleccionada se añade a la selección existente. Si la línea seleccionada ya pertenece a la selección existente, el comando no hace nada.
lk all	Entero largo	0	El comando afecta todos los subniveles (valor por defecto, utilizado si el parámetro se omite).
lk allow wordwrap	Entero largo	14	Propiedad **Retorno de línea** Aplica a: Columna* Valores posibles: • lk no (0) • lk yes (1)
lk auto row height	Entero largo	31	Propiedad **Altura de fila automática**. Aplica a: List box o columna Valores posibles: • lk yes • lk no **4D View Pro únicamente:** esta funcionalidad requiere una licencia 4D View Pro. Para más información, consulte **4D View Pro**.
lk automatic	Entero largo	2	Las columnas se redimensionan automáticamente junto con el list box (Propiedad seleccionada **Autoredimensionamiento de columnas**).
lk background color	Entero largo	1	
lk background color array	Entero largo	1	
lk background color expression	Cadena	22	Propiedad **Expresión color de fondo** para list box de tipo selección, colección o entity selection. Aplica a: List box o columna
lk break row	Entero largo	2	El comando afecta el subnivel al que pertenece la "celda" designada por los parámetros línea y columna. Note que estos parámetros representan los números de línea y de columna en el listbox en modo estándar y no en su representación jerárquica. Si los parámetros línea y columna se omiten, el comando no hace nada.
lk column max width	Entero largo	26	Propiedad **Ancho Máximo** Aplica a: Columna*
lk column min width	Entero largo	25	Propiedad **Ancho mínimo** Aplica a: Columna*
lk column resizable	Entero largo	15	Propiedad **Redimensionable** Aplica a: Columna* Valores posibles: • lk no (0) • lk yes (1)
lk control array	Entero largo	3	
lk detail form name	Cadena	19	Propiedad **Nombre formulario detallado** para la selección de tipo list box Aplica a: List box
lk display	Entero largo	0	Muestra filas extra en blanco agregadas en la parte inferior del list box.
lk display footer	Entero largo	8	0=oculto, 1=se muestra
lk display header	Entero largo	0	0=oculto, 1=se muestra

Constante	Tipo	Valor	Comentario
lk display record	Entero largo	2	Haciendo doble clic en una fila muestra el registro correspondiente en forma detallada definida para el list box. El registro se abre en modo de sólo lectura para que no pueda ser modificado.
lk display type	Entero largo	21	Propiedad **Tipo de visualización** para columnas numéricas Aplica a: Columna* Valores posibles: • lk numeric format (0): muestra valores en formato numérico • lk three states checkbox (1): muestra valores como casillas de selección de tres estados
lk do nothing	Entero largo	0	Hacer doble clic en una línea de list box no dispara ninguna acción automática.
lk double click on row	Entero largo	18	Propiedad **Doble clic en la línea** para los list box de tipo selección Aplica a: List box Valores posibles: • lk do nothing (0): no desencadena ninguna acción automática • lk edit record (1): muestra el registro correspondiente en modo lectura-escritura • lk display record (2): muestra el registro correspondiente en modo de solo lectura
lk edit record	Entero largo	1	Al hacer doble clic en una fila, se visualiza el registro correspondiente en forma detallada definida para el list box. El registro se abre en modo lectura-escritura para que pueda ser modificado.
lk extra rows	Entero largo	13	Propiedad **Ocultar líneas vacías finales** Aplica a: List box Valores posibles: • lk display (0) • lk hide (1)
lk font color	Entero largo	0	
lk font color array	Entero largo	0	
lk font color expression	Cadena	23	Propiedad **Expresión color fuente**para list box de tipo selección, colección o entity selection. Aplica a: List box o columna
lk font style expression	Cadena	24	Propiedad **Expresión estilo** para list boxes de tipo selección, colección o entity selection. Aplica a: List box o columna
lk hide	Entero largo	1	Oculta las líneas vacías finales en la parte inferior del list box.
lk hide selection highlight	Entero largo	16	Propiedad **Ocultar resaltado de selección** Aplica a: List box Valores posibles: • lk no (0) • lk yes (1)
lk highlight set	Cadena	27	Propiedad **Conjunto resaltado** para el list box de tipo selección Aplica a: List box
lk hor scrollbar height	Entero largo	3	Altura en píxeles
lk inherited	Entero largo	-255	
lk last printed row number	Entero largo	0	Devuelve en info el número de la última línea impresa. Permite conocer el número de la próxima línea a imprimir. El número devuelto puede ser mayor al número de líneas efectivamente impresas si el list box contiene las líneas invisibles o si se llama al comando **OBJECT SET SCROLL POSITION** . Por ejemplo, si se han impreso las líneas 1, 18 y 20, info es 20.
lk level	Entero largo	3	El comando afecta todas las líneas de ruptura correspondientes a la columna nivel. Este parámetro designa un número de columna en el list box en modo estándar y no en su representación jerárquica. Si se omite el parámetro nivel, el comando no hace nada.
lk lines	Entero largo	1	La altura designa un número de líneas. 4D calcula la altura de una línea en función de la fuente.
lk manual	Entero largo	0	Las columnas no se redimensionan automáticamente junto con el list box (Propiedad **Redimensionamiento de columnas automático** no seleccionada).
lk mulltiple	Entero largo	2	Se pueden seleccionar varias filas de list box a la vez.
lk multi style	Entero largo	30	Propiedad **Multiestilo** Aplica a: Columna* Valores posibles: • lk no (0)[#/note] • lk yes (1) [#/note]
lk named selection	Cadena	28	Propiedad **Selección temporal** para list box de tipo selección Aplica a: List box
lk no	Entero largo	0	

Constante	Tipo	Valor	Comentario
lk none	Entero largo	0	
lk numeric format	Entero largo	0	Los valores numéricos se muestran en formato numérico en la columna de list box.
lk pixels	Entero largo	0	La altura es un número en píxeles (por defecto)
lk printed height	Entero largo	3	Devuelve en info la altura en píxeles del objeto efectivamente impreso (incluyendo encabezados, líneas, etc.). Recuerde que si el número de líneas a imprimir es inferior a la "capacidad" del list box, su altura se reduce automáticamente.
lk printed rows	Entero largo	1	Devuelve en info el número de líneas efectivamente impresas durante la última llamada al comando **Print object**. Este número incluye las posibles líneas de ruptura añadidas en el caso de un list box jerárquico. Por ejemplo, info es 10 si el list box contiene 20 líneas y las líneas impares están ocultas.
lk printing is over	Entero largo	2	Devuelve en info un booleano indicando si la última línea (visible) del list box se ha impreso. True = la línea se imprimió; De lo contrario, False.
lk remove from selection	Entero largo	2	La línea seleccionada se remueve de la selección existente. Si la línea especificada no pertenece a la selección existente, el comando no hace nada.
lk replace selection	Entero largo	0	La línea seleccionada se convierte en la nueva selección, reemplazando la selección existente. El comando tiene el mismo efecto que un clic de usuario en una línea (sin embargo, el evento On Clicked no se genera). Esta es la acción por defecto (si se omite el parámetro acción).
lk resizing mode	Entero largo	11	Propiedad **Autoredimensionamiento de columnas** Aplica a: List box Valores posibles: • lk manual (0) • lk automatic (2)
lk row height array	Entero largo	4	(Licencia 4D View Pro requerida)
lk row height unit	Entero largo	17	Unidad de la propiedad **Alto de línea** Aplica a: List box Valores posibles: • lk lines (1) • lk pixels (0)
lk row is disabled	Entero largo	2	La línea correspondiente se desactiva. El texto y los controles tales como casillas de selección se atenúan o se ponen grises. Las áreas de entrada de texto editables ya no son editables. Valor por defecto: Activado
lk row is hidden	Entero largo	1	La línea correspondiente está oculta. Ocultar las líneas sólo afecta la visualización del list box. Las líneas ocultas siguen presentes en los arrays y pueden ser manipuladas por programación. Los comandos del lenguaje, más concretamente **LISTBOX Get number of rows** o [#cmd id="971"/], no tienen en cuenta el estado visible/oculto de las líneas. Por ejemplo, en un list box con 10 líneas donde las primeras 9 líneas se ocultan, **LISTBOX Get number of rows** devuelve10. Desde el punto de vista del usuario, la presencia de líneas ocultas en un list box no es visiblemente discernible. Sólo líneas visibles pueden ser seleccionadas (por ejemplo utilizando el comando Seleccionar todo). Valor por defecto: Visible
lk row is not selectable	Entero largo	4	La línea correspondiente no se puede seleccionar (resaltarla no es posible). Las áreas de entrada de texto editables ya no son editables a menos de que la opción "Editar en clic único" esté activada. Los controles tales como casillas de selección y listas siguen siendo funcionales sin embargo. Este parámetro se ignora si el modo de selección del list box es "Ninguno". Valor por defecto: seleccionable
lk row max height	Entero largo	33	
lk row min height	Entero largo	32	
lk selection	Entero largo	1	El comando afecta los subniveles seleccionados.
lk selection mode	Entero largo	10	Propiedad **Modo de selección** Aplica a: List box Valores posibles: • lk none (0) • lk single (1) • lk multiple (2)
lk single	Entero largo	1	Solo se puede seleccionar una fila de list box a la vez.
lk single click edit	Entero largo	29	Propiedad **Editar en clic único** Aplica a: List box Posible valores: • lk no (0) • lk yes (1)

Constante	*Tipo*	*Valor*	*Comentario*
lk sortable	*Entero largo*	*20*	*Propiedad **Ordenable*** *Aplica a: List box* *Valores posibles:* • *lk no (0)* • *lk yes (1)*
lk style array	*Entero largo*	*2*	
lk three states checkbox	*Entero largo*	*1*	*Las columnas con valores numéricos se muestran como casillas de selección de tres estados.*
lk truncate	*Entero largo*	*12*	*Propiedad **Truncar con elipse*** *Aplica a: List box o columna* *Valores posibles:* • *lk without ellipsis (0)* • *lk with ellipsis (1)*
lk ver scrollbar width	*Entero largo*	*5*	*Ancho en píxeles*
lk with ellipsis	*Entero largo*	*1*	*Se muestran puntos suspensivos cuando el contenido del listbox excede el ancho de la columna.*
lk without ellipsis	*Entero largo*	*0*	*No se muestran puntos suspensivos cuando el contenido de las celdas del listbox supera el ancho de la columna.*
lk yes	*Entero largo*	*1*	

Mantenimiento archivo de datos

Constante	Tipo	Valor	Comentario
Compact address table	*Entero largo*	*131072*	*Fuerza la reescritura de la tabla de direcciones de los registros (ralentiza la compactación). Note que en este caso, los números de registro se reescriben. Si sólo pasa esta opción, 4D activa automáticamente la opción "Actualizar registros".*
Create process	*Entero largo*	*32768*	*Cuando se pasa esta opción, la compactación será asincrónica y deberá administrar los resultados utilizando el método de retrollamada (ver a continuación). 4D no mostrará la barra de progreso (es posible hacerlo vía el método de retrollamada). La variable sistema OK toma el valor 1 si el proceso se ha lanzado correctamente y 0 en todos los otros casos. Cuando no se pasa esta opción, la variable OK toma el valor 1 si la compactación se realiza correctamente, de lo contrario 0.*
Do not compact index	*Entero largo*	*2*	
Do not create log file	*Entero largo*	*16384*	*Por lo general, este comando crea un archivo de historial en formato XML (consulte el final de la descripción del comando). Con esta opción, no se creará un archivo de historial.*
Move to replaced files folder	*Entero largo*	*4*	
New file	*Entero largo*	*0*	
New file dialog	*Entero largo*	*1*	
Renumber records	*Entero largo*	*1*	
Timestamp log file name	*Entero largo*	*262144*	*Cuando esta opción se pasa, el nombre del archivo de historial generado contendrá la fecha y hora de su creación; como resultado, no reemplazará cualquier archivo de historial generado anteriormente. Por defecto, si no se pasa esta opción, los nombres de archivos de historial no son marcados con la fecha y hora y cada nuevo archivo generado sustituye al anterior.*
Update records	*Entero largo*	*65536*	*Fuerza la rescritura de todos los registros en función de la definición actual de los campos en la estructura*
Use default folder	*Entero largo*	*1*	*Los datos pasado en parámetro se almacenarán en la carpeta por defecto, llamada nomBase.ExternalData y ubicada al lado del archivo de datos. En este modo, los datos externos son generados por 4D como si estuvieran al interior del archivo de datos.*
Use selected file	*Entero largo*	*2*	
Use structure definition	*Entero largo*	*0*	*4D utilizará los parámetros definidos en la estructura para el almacenamiento del campo (ver manual Modo Diseño). Si pasa de un almacenamiento externo a un almacenamiento interno, el archivo externo no se elimina.*
Verify all	*Entero largo*	*16*	
Verify indexes	*Entero largo*	*8*	*Esta opción verifica la consistencia física de los índices, sin enlace a los datos. Señala llaves inválidas pero no le permite detectar llaves duplicadas (dos índices que apuntan al mismo registro). Este tipo de error sólo puede detectarse con la opción Verificar todos.*
Verify records	*Entero largo*	*4*	

Motor de la base

Constante	Tipo	Valor	Comentario
New record	Entero largo	-3	
No current record	Entero largo	-1	

Nombres de metadatos imágenes

Constante	Tipo	Valor	Comentario
EXIF aperture value	Cadena	EXIF/ApertureValue	Valores posibles: Real (valor APEX)
EXIF brightness value	Cadena	EXIF/BrightnessValue	Valores posibles: Real (valor APEX)
EXIF color space	Cadena	EXIF/ColorSpace	Valores posibles: EXIF Adobe RGB, EXIF s RGB, EXIF Uncalibrated
EXIF components configuration	Cadena	EXIF/ComponentsConfiguration	Valores posibles: EXIF B, EXIF Cb, EXIF Cr, EXIF G, EXIF R, EXIF Unused, EXIF Y
EXIF compressed bits per pixel	Cadena	EXIF/CompressedBitsPerPixel	Valores posibles: Real
EXIF contrast	Cadena	EXIF/Contrast	Valores posibles: EXIF High, EXIF Low, EXIF Normal
EXIF custom rendered	Cadena	EXIF/CustomRendered	Valores posibles: EXIF Normal, EXIF Custom
EXIF date time digitized	Cadena	EXIF/DateTimeDigitized	Valores posibles: Fecha o Texto (Datetime XML)
EXIF date time original	Cadena	EXIF/DateTimeOriginal	Valores posibles: Fecha o Texto (Datetime XML)
EXIF digital zoom ratio	Cadena	EXIF/DigitalZoomRatio	Valores posibles: Real
EXIF EXIF version	Cadena	EXIF/ExifVersion	Valores posibles: array entero (4 valores)
EXIF exposure bias value	Cadena	EXIF/ExposureBiasValue	Valores posibles: Real
EXIF exposure index	Cadena	EXIF/ExposureIndex	Valores posibles: Real
EXIF exposure mode	Cadena	EXIF/ExposureModus	Possible values: EXIF Auto, EXIF Auto Bracket, EXIF Manual
EXIF exposure program	Cadena	EXIF/ExposureProgram	Possible values: EXIF Manual, EXIF Action, EXIF Aperture Priority AE, EXIF Creative, EXIF Landscape, EXIF Exposure Portrait, EXIF Program AE, EXIF Shutter Speed Priority AE
EXIF exposure time	Cadena	EXIF/ExposureTime	Valores posibles: Real
EXIF F number	Cadena	EXIF/FNumber	Valores posibles: Real
EXIF file source	Cadena	EXIF/FileSource	Possible values: EXIF Digital Camera, EXIF Film Scanner, EXIF Reflection Print Scanner
EXIF flash	Cadena	EXIF/Flash	Possible values: EXIF Auto Mode, EXIF Compulsory Flash Firing, EXIF Compulsory Flash Suppression, EXIF Unknown, EXIF Detected, EXIF No Detection Function, EXIF Not Detected, EXIF Reserved
EXIF flash energy	Cadena	EXIF/FlashEnergy	Valores posibles: Real
EXIF flash fired	Cadena	EXIF/Flash/Fired	Possible values: Boolean
EXIF flash function present	Cadena	EXIF/Flash/FunctionPresent	Possible values: Boolean
EXIF flash mode	Cadena	EXIF/Flash/Mode	Possible values: EXIF Auto Mode, EXIF Compulsory Flash Firing, EXIF Compulsory Flash Suppression, EXIF Unknown
EXIF flash pix version	Cadena	EXIF/FlashPixVersion	Possible values: Integer array (4 values)
EXIF flash red eye reduction	Cadena	EXIF/Flash/RedEyeReduction	Possible values: Boolean
EXIF flash return light	Cadena	EXIF/Flash/ReturnLight	Possible values: EXIF Detected, EXIF No Detection Function, EXIF Not Detected, EXIF Reserved
EXIF focal len in 35 mm film	Cadena	EXIF/FocalLenIn35mmFilm	Possible values: Longint
EXIF focal length	Cadena	EXIF/FocalLength	Possible values: Real
EXIF focal plane resolution unit	Cadena	EXIF/FocalPlaneResolutionUnit	Possible values: Longint
EXIF focal plane X resolution	Cadena	EXIF/FocalPlaneXResolution	Valores posibles: Real

Constante	Tipo	Valor	Comentario
EXIF focal plane Y resolution	Cadena	EXIF/FocalPlaneYResolution	Valores posibles: Real
EXIF gain control	Cadena	EXIF/GainControl	Possible values: EXIF High Gain Down, EXIF High Gain Up, EXIF Low Gain Down, EXIF Low Gain Up, EXIF None
EXIF gamma	Cadena	EXIF/Gamma	Valores posibles: Real
EXIF image unique ID	Cadena	EXIF/ImageUniqueID	Valores posibles: Texto
EXIF ISO speed ratings	Cadena	EXIF/ISOSpeedRatings	Possible values: Longint or Longint array
EXIF light source	Cadena	EXIF/LightSource	Possible values: EXIF Unknown, EXIF Cloudy, EXIF Cool White Fluorescent, EXIF D50, EXIF D55, EXIF D65, EXIF D75, EXIF Daylight, EXIF Daylight Fluorescent, EXIF Day White Fluorescent, EXIF Fine Weather, EXIF Flash, EXIF Light Fluorescent, EXIF ISOStudio Tungsten, EXIF Other, EXIF Shade, EXIF Standard Light A, EXIF Standard Light B, EXIF Standard Light C, EXIF Tungsten, EXIF White Fluorescent
EXIF maker note	Cadena	EXIF/MakerNote	Valores posibles: Texto
EXIF max aperture value	Cadena	EXIF/MaxApertureValue	Possible values: Real
EXIF metering mode	Cadena	EXIF/MeteringMode	Possible values: EXIF Other, EXIF Average, EXIF Center Weighted Average, EXIF Multi Segment, EXIF Multi Spot, EXIF Partial, EXIF Spot
EXIF pixel X dimension	Cadena	EXIF/PixelXDimension	Valores posibles: Entero largo (sólo lectura)
EXIF pixel Y dimension	Cadena	EXIF/PixelYDimension	Valores posibles: Entero largo (sólo lectura)
EXIF related sound file	Cadena	EXIF/RelatedSoundFile	Valores posibles: Texto
EXIF saturation	Cadena	EXIF/Saturation	Possible values: EXIF High, EXIF Low, EXIF Normal
EXIF scene capture type	Cadena	EXIF/SceneCaptureType	Possible values: EXIF Scene Landscape, EXIF Night, EXIF Scene Portrait, EXIF Standard
EXIF scene type	Cadena	EXIF/SceneType	Possible values: Longint
EXIF sensing method	Cadena	EXIF/SensingMethod	Possible values: EXIF Color Sequential Area, EXIF Color Sequential Linear, EXIF Not Defined, EXIF One Chip Color Area, EXIF Three Chip Color Area, EXIF Trilinear, EXIF Two Chip Color Area
EXIF sharpness	Cadena	EXIF/Sharpness	Possible values: EXIF High, EXIF Low, EXIF Normal
EXIF shutter speed value	Cadena	EXIF/ShutterSpeedValue	Valores posibles: Real
EXIF spectral sensitivity	Cadena	EXIF/SpectralSensitivity	Valores posibles: Texto
EXIF subject area	Cadena	EXIF/SubjectArea	Possible values: Longint array (2, 3 or 4 values)
EXIF subject dist range	Cadena	EXIF/SubjectDistRange	Possible values: EXIF Unknown, EXIF Close, EXIF Distant, EXIF Macro
EXIF subject distance	Cadena	EXIF/SubjectDistance	Valores posibles: Real
EXIF subject location	Cadena	EXIF/SubjectLocation	Possible values: Longint array (2 values)
EXIF user comment	Cadena	EXIF/UserComment	Valores posibles: Texto
EXIF white balance	Cadena	EXIF/WhiteBalance	Possible values: EXIF Auto, EXIF Manual
GPS altitude	Cadena	GPS/Altitude	Possible values: GPS Above Sea Level, GPS Below Sea Level
GPS altitude ref	Cadena	GPS/AltitudeRef	Possible values: GPS Above Sea Level, GPS Below Sea Level
GPS area information	Cadena	GPS/AreaInformation	Possible values: Text
GPS date time	Cadena	GPS/DateTime	Possible values: Date or Text (XML Datetime)
GPS dest bearing	Cadena	GPS/DestBearing	Possible values: Text (1 character)
GPS dest bearing ref	Cadena	GPS/DestBearingRef	Possible values: Text (1 character)
GPS dest distance	Cadena	GPS/DestDistance	Possible values: Text (1 character)
GPS dest distance ref	Cadena	GPS/DestDistanceRef	Possible values: Text (1 character)

Constante	Tipo	Valor	Comentario
GPS dest latitude	Cadena	GPS/DestLatitude	Valores posibles: Texto
GPS dest latitude deg	Cadena	GPS/DestLatitude/Deg	Possible values: Real
GPS dest latitude dir	Cadena	GPS/DestLatitude/Dir	Possible values: Text (1 character)
GPS dest latitude min	Cadena	GPS/DestLatitude/Min	Possible values: Real
GPS dest latitude sec	Cadena	GPS/DestLatitude/Sec	Possible values: Real
GPS dest longitude	Cadena	GPS/DestLongitude	Valores posibles: Texto
GPS dest longitude deg	Cadena	GPS/DestLongitude/Deg	Possible values: Real
GPS dest longitude dir	Cadena	GPS/DestLongitude/Dir	Possible values: Text (1 character)
GPS dest longitude min	Cadena	GPS/DestLongitude/Min	Possible values: Real
GPS dest longitude sec	Cadena	GPS/DestLongitude/Sec	Possible values: Real
GPS differential	Cadena	GPS/Differential	Possible values: GPS Correction Applied, GPS Correction Not Applied
GPS DOP	Cadena	GPS/DOP	Possible values: Real
GPS img direction	Cadena	GPS/ImgDirection	Possible values: GPS Magnetic north, GPS True north
GPS img direction ref	Cadena	GPS/ImgDirectionRef	Possible values: GPS Magnetic north, GPS True north
GPS latitude	Cadena	GPS/Latitude	Possible values: GPS North, GPS South
GPS latitude deg	Cadena	GPS/Latitude/Deg	Possible values: Real
GPS latitude dir	Cadena	GPS/Latitude/Dir	Possible values: GPS North, GPS South
GPS latitude min	Cadena	GPS/Latitude/Min	Possible values: Real
GPS latitude sec	Cadena	GPS/Latitude/Sec	Valores posibles: Real
GPS longitude	Cadena	GPS/Longitude	Possible values: GPS West, GPS East
GPS longitude deg	Cadena	GPS/Longitude/Deg	Possible values: Real
GPS longitude dir	Cadena	GPS/Longitude/Dir	Possible values: GPS West, GPS East
GPS longitude min	Cadena	GPS/Longitude/Min	Valores posibles: Real
GPS longitude sec	Cadena	GPS/Longitude/Sec	Possible values: Real
GPS map date	Cadena	GPS/MapDate	Possible values: Text
GPS measure mode	Cadena	GPS/MeasureMode	Possible values: GPS 2D, GPS 3D
GPS processing method	Cadena	GPS/ProcessingMethod	Possible values: Text
GPS satellites	Cadena	GPS/Satellites	Valores posibles: Texto
GPS speed	Cadena	GPS/Speed	Possible values: GPS km h, GPS miles h, GPS knots h
GPS speed ref	Cadena	GPS/SpeedRef	Possible values: GPS km h, GPS miles h, GPS knots h
GPS status	Cadena	GPS/Status	Possible values: GPS Measurement in progress, GPS Measurement Interoperability
GPS track	Cadena	GPS/Track	Possible values: Real (0.00..359.99)
GPS track ref	Cadena	GPS/TrackRef	Possible values: Text (1 character)
GPS version ID	Cadena	GPS/VersionID	Possible values: Longint array (4 characters)
IPTC byline	Cadena	IPTC/Byline	Possible values: Text or Text array
IPTC byline title	Cadena	IPTC/BylineTitle	Possible values: Text or Text array
IPTC caption abstract	Cadena	IPTC/CaptionAbstract	Valores posibles: Texto
IPTC category	Cadena	IPTC/Category	Possible values: Text
IPTC city	Cadena	IPTC/City	Possible values: Text
IPTC contact	Cadena	IPTC/Contact	Possible values: Text or Text array

Constante	Tipo	Valor	Comentario
IPTC content location code	Cadena	IPTC/ContentLocationCode	Possible values: Text or Text array
IPTC content location name	Cadena	IPTC/ContentLocationName	Possible values: Text or Text array
IPTC copyright notice	Cadena	IPTC/CopyrightNotice	Possible values: Text
IPTC country primary location code	Cadena	IPTC/CountryPrimaryLocationCode	Valores posibles: Texto
IPTC country primary location name	Cadena	IPTC/CountryPrimaryLocationName	Valores posibles: Texto
IPTC credit	Cadena	IPTC/Credit	Possible values: Text
IPTC date time created	Cadena	IPTC/DateTimeCreated	Possible values: Date or Text (XML Datetime)
IPTC digital creation date time	Cadena	IPTC/DigitalCreationDateTime	Possible values: Date or Text (XML Datetime)
IPTC edit status	Cadena	IPTC/EditStatus	Possible values: Text
IPTC expiration date time	Cadena	IPTC/ExpirationDateTime	Possible values: Date or Text (XML Datetime)
IPTC fixture identifier	Cadena	IPTC/FixtureIdentifier	Valores posibles: Texto
IPTC headline	Cadena	IPTC/Headline	Possible values: Text
IPTC image orientation	Cadena	IPTC/ImageOrientation	Possible values: Text
IPTC image type	Cadena	IPTC/ImageType	Possible values: Text
IPTC keywords	Cadena	IPTC/Keywords	Possible values: Text or Text array
IPTC language identifier	Cadena	IPTC/LanguageIdentifier	Possible values: Text
IPTC object attribute reference	Cadena	IPTC/ObjectAttributeReference	Possible values: Text
IPTC object cycle	Cadena	IPTC/ObjectCycle	Possible values: Text
IPTC object name	Cadena	IPTC/ObjektName	Possible values: Text
IPTC original transmission reference	Cadena	IPTC/OriginalTransmissionReference	Possible values: Text
IPTC originating program	Cadena	IPTC/OriginatingProgram	Possible values: Text
IPTC program version	Cadena	IPTC/ProgramVersion	Possible values: Text
IPTC province state	Cadena	IPTC/ProvinceState	Valores posibles: Texto
IPTC release date time	Cadena	IPTC/ReleaseDateTime	Valores posibles: fecha o texto (Datetime XML)
IPTC scene	Cadena	IPTC/Scene	Possible values: IPTC Action, IPTC Aerial View, IPTC Close Up, IPTC Couple, IPTC Exterior View, IPTC Full Length, IPTC General View, IPTC Group, IPTC Half Length, IPTC Headshot, IPTC Interior View, IPTC Movie Scene, IPTC Night Scene, IPTC Off Beat, IPTC Panoramic View, IPTC Performing, IPTC Posing, IPTC Profile, IPTC Rear View, IPTC Satellite, IPTC Single, IPTC Symbolic, IPTC Two, IPTC Under Water
IPTC source	Cadena	IPTC/Source	Possible values: Text
IPTC special instructions	Cadena	IPTC/SpecialInstructions	Possible values: Text
IPTC star rating	Cadena	IPTC/StarRating	Possible values: Longint
IPTC sub location	Cadena	IPTC/SubLocation	Possible values: Text
IPTC subject reference	Cadena	IPTC/SubjectReference	Possible values: Longint or Longint array

Constante	Tipo	Valor	Comentario
IPTC supplemental category	Cadena	IPTC/SupplementalCategory	Possible values: Text or Text array
IPTC urgency	Cadena	IPTC/Urgency	Possible values: Longint
IPTC writer editor	Cadena	IPTC/WriterEditor	Possible values: Text or Text array
TIFF artist	Cadena	TIFF/Artist	Valores posibles: Texto
TIFF compression	Cadena	TIFF/Compression	Possible values: TIFF Adobe Deflate, TIFF CCIRLEW, TIFF CCITT1D, TIFF DCS, TIFF Deflate, TIFF Epson ERF, TIFF IT8BL, TIFF IT8CTPAD, TIFF IT8LW, TIFF IT8MP, TIFF JBIG, TIFF JBIGB&W, TIFF JBIGColor, TIFF JPEG, TIFF JPEG2000, TIFF JPEGThumbs Only, TIFF Kodak262, TIFF Kodak DCR, TIFF Kodak KDC, TIFF LZW, TIFF MDIBinary Level Codec, TIFF MDIProgressive Transform Codec, TIFF MDIVector, TIFF Next, TIFF Nikon NEF, TIFF Pack Bits, TIFF Pentax PEF, TIFF Pixar Film, TIFF Pixar Log, TIFF SGILog, TIFF SGILog24, TIFF Sony ARW, TIFF T4Group3Fax, TIFF T6Group4Fax, TIFF Thunderscan, TIFF Uncompressed
TIFF copyright	Cadena	TIFF/Copyright	Possible values: Text
TIFF date time	Cadena	TIFF/DateTime	Possible values: Date or Text (XML Datetime)
TIFF document name	Cadena	TIFF/DocumentName	Possible values: Text
TIFF host computer	Cadena	TIFF/HostComputer	Valores posibles: Texto
TIFF image description	Cadena	TIFF/ImageDescription	Valores posibles: Texto
TIFF make	Cadena	TIFF/Make	Valores posibles: Texto
TIFF model	Cadena	TIFF/Model	Valores posibles: Texto
TIFF orientation	Cadena	TIFF/Orientation	Valores posibles: TIFF Horizontal, TIFF Mirror Horizontal, TIFF Mirror Horizontal And Rotate270CW, TIFF Mirror Horizontal And Rotate90CW, TIFF Mirror Vertical, TIFF Rotate180, TIFF Rotate270CW, TIFF Rotate90CW
TIFF photometric interpretation	Cadena	TIFF/PhotometricInterpretation	Valores posibles: TIFF Black Is Zero, TIFF CIELab, TIFF CMYK, TIFF Color Filter Array, TIFF ICCLab, TIFF ITULab, TIFF Linear Raw, TIFF Pixar Log L, TIFF Pixar Log Luv, TIFF RGB, TIFF RGBPalette, TIFF Transparency Mask, TIFF White Is Zero, TIFF YCb Cr
TIFF resolution unit	Cadena	TIFF/ResolutionUnit	Valores posibles: TIFF CM, TIFF Inches, TIFF MM, TIFF None, TIFF UM
TIFF software	Cadena	TIFF/Software	Valores posibles: Texto
TIFF xResolution	Cadena	TIFF/XResolution	Valores posibles: Real
TIFF yResolution	Cadena	TIFF/YResolution	Valores posibles: Real

Números de puerto TCP

Constante	*Tipo*	*Valor*	*Comentario*
TCP Authentication	*Entero largo*	*113*	
TCP DNS	*Entero largo*	*53*	
TCP Finger	*Entero largo*	*79*	
TCP FTP Control	*Entero largo*	*21*	
TCP FTP Data	*Entero largo*	*20*	
TCP Gopher	*Entero largo*	*70*	
TCP HTTP WWW	*Entero largo*	*80*	
TCP IMAP3	*Entero largo*	*220*	
TCP Kerberos	*Entero largo*	*88*	
TCP KLogin	*Entero largo*	*543*	
TCP Nickname	*Entero largo*	*43*	
TCP NNTP	*Entero largo*	*119*	
TCP NTalk	*Entero largo*	*518*	
TCP NTP	*Entero largo*	*123*	
TCP PMCP	*Entero largo*	*1643*	
TCP PMD	*Entero largo*	*1642*	
TCP POP3	*Entero largo*	*110*	
TCP Printer	*Entero largo*	*515*	
TCP RADACCT	*Entero largo*	*1646*	
TCP RADIUS	*Entero largo*	*1645*	
TCP Remote Cmd	*Entero largo*	*514*	
TCP Remote Exec	*Entero largo*	*512*	
TCP Remote Login	*Entero largo*	*513*	
TCP Router	*Entero largo*	*520*	
TCP SMTP	*Entero largo*	*25*	
TCP SNMP	*Entero largo*	*161*	
TCP SNMPTRAP	*Entero largo*	*162*	
TCP SUN RPC	*Entero largo*	*111*	
TCP Talk	*Entero largo*	*517*	
TCP Telnet	*Entero largo*	*23*	
TCP TFTP	*Entero largo*	*69*	
TCP UUCP	*Entero largo*	*540*	
TCP UUCP RLOGIN	*Entero largo*	*541*	

Objetos de formulario (Acceso)

Constante	Tipo	Valor	Comentario
Form all pages	*Entero largo*	*2*	*Devuelve todos los objetos de todas las páginas, excluyendo los objetos heredados*
Form current page	*Entero largo*	*1*	*Devuelve todos los objetos de la página actual, incluyendo la página 0, pero excluyendo los objetos heredados*
Form inherited	*Entero largo*	*4*	*Devuelve sólo los objetos heredados*
Object current	*Entero largo*	*0*	
Object first in entry order	*Cadena*	*;FirstObject*	
Object named	*Entero largo*	*3*	
Object subform container	*Entero largo*	*2*	
Object with focus	*Entero largo*	*1*	

Objetos y colecciones

Constante	Tipo	Valor	Comentario
ck ascending	Entero largo	0	Los elementos se ordenan de forma ascendente (predeterminado)
ck descending	Entero largo	1	Los elementos se ordenan en orden descendente
ck diacritical	Entero largo	8	Ejecutar una evaluación a diacrítica
ck disable wildchar	Entero largo	16	
ck ignore null or empty	Entero largo	1	Ignorar valores nulos y cadenas vacías en el resultado
ck keep empty strings	Entero largo	2	
ck keep null	Entero largo	1	Conservar las propiedades null o undefined en el resultado
ck resolve pointers	Entero largo	1	Resolver punteros en la copia
dk auto merge	Entero largo	32	Activa el modo "fusión" automática para el método **entity.save( )**
dk diacritical	Entero largo	8	Efectuar una evaluación diacrítica
dk distinct values	Entero largo	1	Considere solo los atributos de entidad con valores no repetidos (opción para el método **entitySelection.count( )**)
dk force drop if stamp changed	Entero largo	2	Fuerza la supresión incluso si el stamp ha cambiado (opción para el método **entity.drop( )**)
dk keep ordered	Entero largo	1	Opción para mantener el orden de la colección de origen en la nueva colección
dk key as string	Entero largo	1	El método **entity.getKey( )** devuelve el valor de la llave primaria como cadena (texto)
dk non ordered	Entero largo	0	Opción para crear una nueva colección no ordenada
dk reload if stamp changed	Entero largo	1	Si el stamp de la entidad ha cambiado, vuelva a cargar la entidad antes de realizar el proceso de bloqueo (opción para el método **entity.lock( )**)
dk status automerge failed	Entero largo	6	(Solo si se usa la opción dk auto merge) La opción de combinación automática falló al guardar la entidad. **statusText asociado**: "Auto merge failed"
dk status entity does not exist anymore	Entero largo	5	La entidad ya no existe en los datos. Este error puede ocurrir en los siguientes casos: • la entidad ha sido suprimida (el stamp ha cambiado y el espacio de memoria ahora está liberado) • la entidad ha sido suprimida y reemplazada por otra con una llave primaria diferente (el stamp ha cambiado y una nueva entidad ahora ocupa el espacio de memoria). Cuando se usa **entity.drop( )**, este error puede devolverse cuando se usa la opción dk force drop if stamp changed. Al usar **entity.lock( )**, este error puede devolverse cuando se usa la opción dk reload if stamp changed **statusText asociado**: "La entidad ya no existe"
dk status locked	Entero largo	3	La entidad está bloqueada por un bloqueo pesimista. **statusText asociado**: "Ya bloqueado" ("Already locked")
dk status serious error	Entero largo	4	Un error grave es un error de bajo nivel de la base de datos (por ejemplo, llave duplicada), un error de hardware, etc. **statusText asociado**: "Otro error" ("Other error")
dk status stamp has changed	Entero largo	2	El valor de stamp interno de la entidad no coincide con el de la entidad almacenada en los datos (bloqueo optimista). • con **entity.save( )**: error solo si no se utiliza la opción dk auto merge • con **entity.drop( )**: error solo si no se usa la opción dk force drop if stamp changed • con **entity.lock( )**: error solo si no se usa la opción dk reload if stamp changed **statusText asociado**: "El stamp ha cambiado" ("Stamp has changed")
dk stop dropping on first error	Entero largo	8	La acción de supresión se detiene en la primera entidad no eliminable (opción para el método **entitySelection.drop( )**)
dk with primary key	Entero largo	1	Agregue la llave primaria en la colección u objeto extraído (opción para los métodos **entitySelection.toCollection( )** y **entity.toObject( )**)
dk with stamp	Entero largo	2	Agregue el stamp en la colección u objeto extraído (opción para los métodos **entitySelection.toCollection( )** y **entity.toObject( )**)

Opciones de impresión

Constante	Tipo	Valor	Comentario
Color option	Entero largo	8	(Windows únicamente) valor1 únicamente: código que especifica el modo para el manejo del color: 1=Blanco y negro (monocromo), 2=Color. **Versiones 64 bits**: esta opción no está disponible en versiones 4D de 64 bits (obsoleto)
Destination option	Entero largo	9	valor1: código que especifica el tipo de destino de la impresión: 1=Impresora, 2=Archivo (PC)/PS (Mac), 3=Archivo PDF, 5=Pantalla (opción del driver OS X). Si valor1 es diferente de 1 o 5, valor2 contiene un nombre de ruta para el documento resultante. Esta ruta se utilizará hasta que se especifique otra ruta. Si un archivo con el mismo nombre ya existe en el lugar de destino, será sustituido. Con **GET PRINT OPTION**, si el valor actual no está en la lista predefinida, valor1 contiene -1 y la variable sistema OK toma el valor 1. Si ocurre un error, valor1 y la variable sistema OK toman el valor 0. **Nota**: en Windows, puede definir el destino de impresión 3 (archivo PDF) cuando el driver PDF Creator ha sido instalado. Cuando se pasan los valores (9;3;ruta), 4D inicia automáticamente una impresión PDF "silenciosa" que tiene en cuenta los códigos de opción pasados (note que si pasa una cadena vacía en valor2 u omite este parámetro, aparece el diálogo de guardar archivo en el momento de la impresión.) Después de la impresión, los parámetros actuales se restauran. Esto simplifica el control de las impresiones PDF por 4D y permite escribir código multiplataforma. Cuando los valores (9;3;ruta) no se transmiten, la impresión no es controlada por 4D y los eventuales códigos de opciones de PDF Creator se ignoran.
Double sided option	Entero largo	11	(Windows únicamente) valor1: 0=Un solo lado o estándar, 1=Doble cara. Si valor1=1, valor2 contiene la unión: 0=Izquierda (valor predeterminado), 1=Unión superior. **Nota**: esta opción sólo se puede utilizar en Windows.
Generic PDF driver	Cadena	_4d_pdf_printer	**Nota:** esta funcionalidad no está disponible en las versiones 32 bits de 4D. • En OS X, declara el driver predeterminado como impresora actual. Este driver no es visible y no está en la lista devuelta por el comando **PRINTERS LIST**. la ruta de acceso al documento PDF se debe definir utilizando el comando **SET PRINT OPTION**, si no, se devuelve el error 3107. • En Windows, declara el driver PDF de Windows (llamado "Microsoft Print to PDF") como impresora actual. Esta constante está disponible en Windows 10 únicamente, cuando está instalada la opción PDF. Con otras versiones de Windows, o cuando no hay ningún driver PDF disponible, el comando no hace nada y la variable OK toma el valor 0.
Hide printing progress option	Entero largo	14	(Mac únicamente) valor1 únicamente: 1=ocultar ventanas de progreso, 0= mostrar las ventanas de progreso de impresión (por defecto). Esta es una opción particularmente útil en el caso de impresión PDF en OS X. **Nota**: ya existe una opción de progreso de impresión accesible vía el cuadro de diálogo Propiedades de la base (página Interfaz). Sin embargo, se aplica globalmente a la aplicación y no oculta todas las ventanas bajo OS X.
Legacy printing layer option	Entero largo	16	(Versiones 4D 64 bits para Windows únicamente) valor1 únicamente: 1=seleccionar la antigua capa de impresión GDI para todos los trabajos de impresión subsiguientes. 0=seleccionar la capa de impresión D2D (por defecto). **Versiones 64 bits:** este selector sólo es compatible con las aplicaciones 4D 64 bits mono usuario en Windows; se ignora en otras plataformas. Está destinado principalmente para permitir plug-ins de antigua generación imprimir dentro de tareas de impresión 4D.
Mac spool file format option	Entero largo	13	(Mac únicamente) valor1 únicamente: 0=impresión en modo PDF (valor por defecto) 1 = impresión en modo PostScript. **Notas:** - Esta opción no tiene efecto en Windows.. - En OS X, la impresión se realiza en formato PDF de forma predeterminada. Sin embargo, el driver de impresión PDF no es compatible con imágenes PICT con información PostScript encapsulada, estas imágenes se generan, particularmente, por los softwares de dibujo vectorial. Para evitar este problema, esta opción le permite modificar el modo de impresión a utilizar en OS X para la sesión actual. Tenga en cuenta la impresión en modo PostScript puede conducir a efectos secundarios no deseados. **Versiones 64 bits:** esta opción no es compatible; Es reemplazada por la opción Driver PDF générique del comando **SET CURRENT PRINTER**.
Number of copies option	Entero largo	4	valor1 únicamente: número de copias a imprimir.
Orientation option	Entero largo	2	valor1 únicamente: 1=Retrato, 2=Paisaje. Si se utiliza una opción de orientación diferente, **GET PRINT OPTION**devuelve 0 en valor1. **Versiones 64 bits**: esta opción se puede llamar desde una tarea de impresión, lo que significa que puede cambiar de vertical a horizontal, o viceversa, durante el mismo trabajo de impresión.
Page range option	Entero largo	15	valor1=primera página a imprimir (valor por defecto 1) y (opcional) valor2=número de la última página a imprimir (valor por defecto -1 = fin del documento).
Page setup dialog	Entero largo	1	Visualización del diálogo de configuración de página

Constante	Tipo	Valor	Comentario
Paper option	Entero largo	1	Si sólo utiliza valor1, que contiene el nombre del papel. Si se utilizan los dos parámetros, valor1 contiene el ancho del papel y valor2 contiene el alto del papel. El ancho y el alto se expresan en píxeles de pantalla. Utilice el comando **PRINT OPTION VALUES** para obtener el nombre, el alto y el ancho de todos los formatos de papel que ofrece la impresora.
Paper source option	Entero largo	5	(Windows únicamente) valor1 únicamente: número correspondiente al índice, en el array de bandejas devuelto por el comando **PRINT OPTION VALUES**, de la bandeja de papel a utilizar. Esta opción sólo se puede utilizar en Windows.
PDFCreator Printer name	Cadena	PDFCreator	Visualización de la caja de impresión
Print dialog	Entero largo	2	Visualización de la caja de diálogo de Impresión
Scale option	Entero largo	3	valor1 únicamente: valor de la escala en porcentaje. Tenga cuidado, algunas impresoras no permiten modificar la escala. Si pasa un valor no válido, la propiedad se reinicia al 100% en el momento de la impresión.
Spooler document name option	Entero largo	12	valor1 únicamente: nombre del documento de impresión actual, que aparece en la lista de documentos de la cola de impresión. El nombre definido para esta instrucción se utilizará para todos los documentos de impresión de la sesión hasta que un nuevo nombre o una cadena vacía no se pase. Para utilizar o restablecer el funcionamiento normal (usando el nombre del método en el caso de un método, el nombre de la tabla para un registro, etc.), pase una cadena vacía en valor1.

Parámetro de formulario

Constante	*Tipo*	*Valor*	*Comentario*
Multiple selection	*Entero largo*	*2*	*El usuario puede seleccionar varios registros al tiempo. Para seleccionar registros contiguos, haga clic en el primer registro a seleccionar, luego presione la tecla **Mayús** antes de hacer clic en el último registro a incluir en la selección. Para seleccionar registros no adyacentes, haga clic en cada registro por separado mientras presiona la tecla **Ctrl** (Windows) o **Comando** (Mac OS).*
No selection	*Entero largo*	*0*	*No es posible seleccionar un registro en la lista*
NonInverted objects	*Entero largo*	*0*	
Single selection	*Entero largo*	*1*	*Sólo es posible seleccionar un registro a la vez*

Parámetros de la base

Constante	Tipo	Valor	Comentario
_o_4D Local mode scheduler	Entero largo	10	**** Este selector es obsoleto y no debe utilizarse ****
_o_4D Remote mode scheduler	Entero largo	12	**** Este selector es obsoleto y no debe utilizarse ***
_o_4D Server scheduler	Entero largo	11	**** Este selector es obsoleto y no debe utilizarse ***
_o_Client IP address to listen	Entero largo	23	**** Selector inactivo, utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** ****
_o_Database cache size	Entero largo	9	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: - **Descripción**: constante obsoleta (Conservada por razones de compatibilidad únicamente). Ahora se recomienda utilizar el comando **Get cache size**.
_o_IP Address to listen	Entero largo	16	**** Selector inactivo, utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** ****
_o_Real display precision	Entero largo	32	**** Selector desactivado ****
_o_Web conversion mode	Entero largo	8	**** Selector desactivado ****
_o_Web Log recording	Entero largo	29	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
4D Remote mode timeout	Entero largo	14	**Alcance** (antigua capa de red únicamente): aplicación 4D si valor positivo **Se conserva entre dos sesiones**: sí si valor positivo **Descripción**: a utilizar en casos muy específicos. Valor del timeout otorgado por el equipo 4D remoto a la máquina 4D Server. Por defecto, este valor se define en la página "Cliente-Servidor/Configuración" de la caja de diálogo de Preferencias en el equipo remoto. El selector Timeout 4D mode distant no se tiene en cuenta si utiliza la antigua capa de red. Con la capa 4D ServerNet activada, se ignora: esta configuración es administrada por el selector Timeout 4D Server (13).
4D Server log recording	Entero largo	28	**Alcance**: 4D Server, 4D remoto **Se conserva entre dos sesiones**: no **Valores posibles**: 0 ó de 1 a X (0 = no grabar, 1 a X = número secuencial, añadido al nombre del archivo). **Descripción**: inicia o detiene la grabación de las peticiones estándar recibidas por 4D Server (excluyendo las peticiones web). Por defecto, el valor es 0 (no se graban las peticiones). 4D Server le permite grabar cada petición recibida por el equipo servidor en un archivo de historial. Cuando este mecanismo está activo, el archivo de historial se crea junto al archivo de estructura de la base. Su nombre es "4DRequestsLog_X," donde X es el número secuencial del historial. Una vez el archivo alcanza un tamaño de 10 MB, se cierra y se genera un nuevo archivo, con un número secuencial incrementado. Si existe un archivo con el mismo nombre, se reemplaza directamente. Puede definir el número de inicio de la secuencia utilizando el parámetro valor. Este archivo texto almacena en formato tabulado simple diferente información sobre cada petición: hora, número de proceso, usuario, tamaño de la petición, duración del proceso, etc. Esta información puede ser útil particularmente durante la fase de afinamiento de la aplicación o con fines estadísticos. Por ejemplo puede importarse, en un software de hoja de cálculo para procesarse.

Constante	Tipo	Valor	Comentario
4D Server timeout	Entero largo	13	**Alcance**: aplicación 4D si valor positivo **Se conserva entre dos sesiones**: sí si valor positivo **Valores posibles**: 0 -> 32 767 **Descripción**: valor del tiempo de espera antes de desconexión (timeout) de 4D Server a los equipos clientes. Por defecto, este valor se define en la página "Cliente-Servidor/Configuración" de la caja de diálogo Preferencias en el equipo servidor. El timeout del servidor define el periodo máximo de no respuesta del cliente "autorizado", por ejemplo si realiza una operación de bloqueo. Al terminar esta periodo, 4D Server desconecta al cliente. El selector 4D Server Timeout le permite asignar en el parámetro valor un nuevo timeout, expresado en minutos. Esta funcionalidad es particularmente útil para aumentar el valor del timeout antes de la ejecución en el equipo cliente de una operación de larga duración, tal como la impresión de un gran número de páginas, la cual puede causar un timeout inesperado. Tiene dos opciones: • Si pasa un valor **positivo** en el parámetro valor, define un timeout global y permanente: el nuevo valor se aplica a todos los procesos y se almacena en las Preferencias de la aplicación 4D (equivalente a cambiar en el diálogo Preferencias). • Si pasa un valor **negativo** en el parámetro valor, define un timeout lobal y temporal: el nuevo valor se aplica únicamente a los procesos llamantes (los otros procesos conservan los valores por defecto) y se restaura al valor por defecto tan pronto como el servidor recibe una señal de actividad del cliente, por ejemplo, cuando la operación termina. Esta opción es muy útil para administrar operaciones largas iniciadas por plug-ins 4D. Para definir una conexión "Sin timeout", pase 0 en valor. Ver el ejemplo 1.
Auto synchro resources folder	Entero largo	48	**Alcance**: equipo 4D remoto **Se conserva entre dos sesiones**: no **Valores posibles**: 0 (sin sincronización), 1 (auto sincronización) ó 2 (preguntar). **Descripción**: modo de sincronización dinámico de la carpeta Resources del equipo cliente 4D que ejecuta el comando con el servidor. Cuando el contenido de la carpeta Resources en el servidor se ha modificado o un usuario ha solicitado la sincronización (por ejemplo vía el explorador de recursos o siguiendo la ejecución del comando **NOTIFY RESOURCES FOLDER MODIFICATION**), el servidor notifica a los equipos cliente conectados. Tres modos de sincronización son posibles del lado del cliente. El selector Auto Synchro Resources Folder se utiliza para especificar el modo a utilizar por el equipo cliente para la sesión actual: • 0 (valor por defecto): sin sincronización dinámica (la petición de sincronización se ignora) • 1: sincronización dinámica automática • 2: visualización de una caja de diálogo en los equipos clientes, con la posibilidad de efectuar o rechazar la sincronización. El modo de sincronización también puede definirse globalmente en las Preferencias de la aplicación.
Cache flush periodicity	Entero largo	95	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones:** no **Valores posibles:** entero largo > 1 (segundos) **Descripción**: obtiene o establece la periodicidad del vaciado de la caché, expresado en segundos. La modificación de este valor prevalece sobre la opción **Vaciar caché cada X segundos** en **Página Base de datos/Memoria** de la configuración de la base para la sesión (que no se almacena en las Propiedades de la base).
Cache unload minimum size	Entero largo	66	**Alcance**: Aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: Entero largo positivo > 1. **Descripción**: tamaño mínimo de memoria a liberar del caché de la base de datos cuando el motor necesita hacer espacio para ubicar un objeto (valor en bytes). El propósito de este selector es reducir el número de liberaciones de datos de la caché con el fin de obtener un mejor rendimiento. Puede hacer variar este parámetro en función del tamaño de la caché y del de los bloques de datos manipulados en su base. Por defecto, si este selector no se utiliza, 4D descarga mínimo 10% de la caché en caso de que se necesite espacio.Alcance: Aplicación 4D Se conserva entre dos sesiones: No Valores posibles: Entero largo positivo > 1. Descripción: tamaño mínimo de memoria a liberar del caché de la base de datos cuando el motor necesita hacer espacio para ubicar un objeto (valor en bytes). El propósito de este selector es reducir el número de liberaciones de datos de la caché con el fin de obtener un mejor rendimiento. Puede hacer variar este parámetro en función del tamaño de la caché y del de los bloques de datos manipulados en su base. Por defecto, si este selector no se utiliza, 4D descarga mínimo 10% de la caché en caso de que se necesite espacio.
Character set	Entero largo	17	**Alcance:** 4D local, 4D Server **Se conserva entre dos sesiones:** sí **Descripción:** Constante obsoleta (se conserva por compatibilidad únicamente). Ahora recomendamos utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.

Constante	Tipo	Valor	Comentario
Circular log limitation	*Entero largo*	*90*	***Alcance**: 4D local, 4D Server.* ***Se conserva entre dos sesiones:** no* ***Valores posibles**: todo valor entero, 0 = conservar todos los registros* ***Descripción**: número máximo de archivos a conservar en rotación para cada tipo de registro. Por defecto, todos los archivos se conservan. Si pasa un valor X, solo los X archivos más recientes se conservan, el más antiguo se borra automáticamente cuando se crea uno nuevo. Este ajuste se aplica a cada uno de los siguientes archivos de registro: registros de peticiones (selectores 28 y 45), registro de depuración (selector 34), registro de eventos (selector 79), así como el historial de peticiones web y el historial de depuración Web (selectores 29 y 84 del comando **WEB SET OPTION**).*
Client character set	*Entero largo*	*24*	***Alcance**: todos los equipos 4D remotos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: ver selector 17* ***Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir los valores sólo para algunos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.*
Client HTTPS port ID	*Entero largo*	*40*	***Alcance**: todos los equipos 4D remotos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: 0 a 65535* ***Descripción**: número de puerto TCP utilizado por los servidores web de los equipos clientes para conexiones seguras vía SSL (protocolo HTTPS). Por defecto, el valor es 443 (valor estándar).* *Este selector puede utilizarse para modificar por programación el puerto TCP utilizado por los servidores web de los equipos clientes para las conexiones seguras vía SSL (protocolo HTTPS). Por defecto, el valor es 443 (valor estándar).* *Este selector funciona exactamente igual que el selector 39; sin embargo, aplica a todos los equipos 4D remotos utilizados como servidores web. Si quiere modificar el valor de ciertos equipos clientes únicamente, utilice la caja de diálogo de Preferencias de 4D remoto.*
Client log recording	*Entero largo*	*45*	***Alcance**: equipo 4D remoto* ***Se conserva entre dos sesiones**: no* ***Valores posibles**: 0 ó de 1 a X (0 = no grabar, 1 a X = número secuencial, asociado al nombre del archivo).* ***Descripción**: inicia o detiene la grabación de peticiones estándar efectuadas por el equipo cliente 4D que ejecutó el comando (excluyendo las peticiones web). Por defecto, el valor es 0 (no se graban las peticiones).* *4D le permite registrar el historial de peticiones realizadas por el equipo cliente. Cuando este mecanismo se activa, se crean dos archivos en el equipo cliente, en la subcarpeta Logs de la carpeta local de la base. Son llamados 4DRequestsLog_X y 4DRequestsLog_ProcessInfo_X, donde X es el número secuencial del historial. Una vez el archivo 4DRequestsLog alcanza un tamaño de 10 MB, se cierra y se genera uno nuevo, con un número secuencial incrementado. Si ya existe un archivo con el mismo nombre, se reemplaza directamente. Puede definir el número de inicio para la secuencia utilizando el parámetro valor.* *Estos archivos texto almacenan en formato tabulado simple diferente información relacionada con cada petición: hora, número de proceso, tamaño de la petición, duración del proceso, etc. Esta información es particularmente útil durante la fase de desarrollo de la aplicación o con fines estadísticos.*
Client max concurrent Web proc	*Entero largo*	*25*	***Alcance**: todos los equipos 4D remotos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: ver selector 18* ***Descripción**: permite especificar esta parámetro para las máquinas 4D remotas utilizadas como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir este valor sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.*
Client Max Web requests size	*Entero largo*	*21*	***Alcance**: todos los equipos 4D remotos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: ver selector 27* ***Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.*
Client maximum Web process	*Entero largo*	*20*	***Alcance**: todos los equipos 4D remotos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: ver selector 7* ***Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.*
Client minimum Web process	*Entero largo*	*19*	***Alcance**: todos los equipos 4D remotos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: ver selector 6* ***Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.*

Constante	Tipo	Valor	Comentario
Client port ID	Entero largo	22	***Alcance**: todos los equipos 4D remotos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: ver selector 15* ***Descripción**: permite especificar este parámetro para todos los equipos 4D remotos utilizados como servidores web. Los valores definidos utilizando estos selectores se aplican a todos los equipos remotos utilizados como servidores web. Si quiere definir valores sólo para ciertos equipos remotos, utilice la caja de diálogo de Preferencias de 4D en modo remoto.*
Client Server port ID	Entero largo	35	***Alcance**: base de datos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: 0 a 65535* ***Descripción**: número de puerto TCP donde el servidor 4D publica la base de datos (para conexión remota 4D). Por defecto, el valor es 19813.* *La personalización de este valor permite utilizar varias aplicaciones 4D cliente-servidor en la misma máquina con el protocolo TCP; en este caso, debe indicar un número de puerto diferente para cada aplicación.* *El valor se guarda en el archivo de estructura de la base. Puede definirse con 4D en modo local pero sólo se tiene en cuenta en configuración cliente servidor.* *Cuando modifica este valor, es necesario reiniciar el equipo servidor para que el nuevo valor sea tenido en cuenta.*
Client Web log recording	Entero largo	30	***Alcance**: todos los equipos 4D remotos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: 0 = No grabar (por defecto), 1 = Registrar en formato CLF, 2 = Registrar en formato DLF, 3 = Registrar en formato ELF, 4 = Registrar en formato WLF.* ***Descripción**: inicia o detiene la grabación de las peticiones web recibidas por los servidores web de todos los equipos cliente. Por defecto, el valor es 0 (no se graban las peticiones).* *El funcionamiento de este selector es idéntico al del selector 29; sin embargo, aplica a todos los equipos 4D remotos utilizados como servidores web. El archivo "logweb.txt", en este caso, automáticamente ubicado en la subcarpeta Logs de la base 4D remota (carpeta de caché). Si quiere definir los valores únicamente para ciertos equipos cliente, utilice la caja de diálogo de Preferencias de 4D en modo remoto.*
Date type	Entero largo	2	*Selector tipo de datos para Dates inside objects*
Dates inside objects	Entero largo	85	***Alcance**: Proceso actual* ***Se conserva entre dos sesiones:** No* ***Valores posibles**: String type without time zone (0), String type with time zone (1), Date type (2) (por defecto)* ***Descripción**: define la forma en que se almacenan las fechas dentro de los objetos, así como también cómo se importan / exportan en JSON.* *Cuando el valor del selector es Date type (valor predeterminado para las bases creadas con 4D v17 y superior), las fechas 4D se almacenan con el tipo de fecha dentro de los objetos, con respecto a la configuración de fecha local. Cuando se convierte a formato JSON, los atributos de fecha se convertirán en cadenas que no incluyen un tiempo. (**Nota:** esta configuración se puede definir mediante la opción "Utilizar tipo de fecha en lugar del formato de fecha ISO en objetos" que se encuentra en **Página Compatibilidad** de la configuración de la base).* *Si pasa String type with time zone en este selector, convertirá las fechas 4D en cadenas ISO y tendrá en cuenta la zona horaria local. Por ejemplo, la conversión de la fecha 23/08/2013 le da "2013-08-22T22: 00: 000Z" en formato JSON cuando la operación se realiza en Francia durante el horario de verano (GMT+ 2). Este principio se ajusta al funcionamiento estándar de JavaScript. Esto puede ser una fuente de errores cuando desea enviar valores de fecha JSON a alguien en un huso horario diferente. Por ejemplo, cuando exporta una tabla usando **Selection to JSON** en Francia que se debe reimportar en los EE. UU. utilizando **JSON TO SELECTION**. Dado que las fechas se vuelven a interpretar en cada zona horaria, los valores almacenados en la base de datos serán diferentes. En este caso, puede modificar el modo de conversión de las fechas para que no tengan en cuenta la zona horaria pasando String type without time zone en este selector. La conversión de la fecha 23/08/2013 le dará "2013-08-23T00: 00: 00Z" en todos los casos.*

Constante	Tipo	Valor	Comentario

Debug log recording	*Entero largo*	*34*	***Alcance**: Aplicación 4D*
			***Se conserva entre dos sesiones**: No*
			***Descripción**: inicia o detiene la grabación secuencial de los eventos a nivel de programación de 4D en el archivo 4DDebugLog, que se ubica automáticamente en la subcarpeta Logs de la base de datos, junto al archivo de estructura. Un nuevo formato texto tabulado, más compacto se utiliza en el archivo de registro de eventos "4DDebugLog [_n].txt" a partir de 4D v14 (donde _n es el número de segmento del archivo).*
			***Valores posibles**: Entero largo contiene un campo de bits: valor = bit1(1)+bit2(2)+bit3(4)+bit4(8)+…).*
			- Bit 1 (valor 1) permite activar el archivo (note que cualquier otro valor no nulo también lo activará)
			- Bit 2 (valor 2) permite solicitar los parámetros de llamada a los métodos y comandos.
			- Bit 3 (valor 4) permite activar el nuevo formato tabulado.
			- Bit 4 (valor 8) permite desactivar la escritura inmediata de cada operación en el disco (activado por defecto). La escritura inmediata es menor rápida y más eficaz por ejemplo para buscar las causas de un fallo. Si desactiva este modo, el contenido del archivo será más compacto y se generará más rápidamente.
			- Bit 5 (valor 16) desactiva el registro de llamadas de plug-ins (activado por defecto).
			En el formato no tabulado (anterior), los tiempos de ejecución se expresaban en milisegundos y el valor "< ms" se muestra si una operación se ejecuta en menos de un milisegundo.
			En el nuevo formato tabulado, los tiempos de ejecución se expresan en microsegundos.
			Ejemplos:
			SET DATABASE PARAMETER (34;1) // activa el archivo modo v13 sin los parámetros, con las duraciones
			SET DATABASE PARAMETER (34;2) // activa el archivo modo v13 con los parámetros y las duraciones
			SET DATABASE PARAMETER (34;2+4) // activa el archivo al formato v14 con los parámetros y las duraciones
			SET DATABASE PARAMETER (34;0) // desactiva el archivo
			Para evitar que el archivo registre demasiada información, puede restringir los comandos 4D a examinar con el selector 80, Log Command list.
			Esta opción se puede activar para todo tipo de aplicación 4D (4D todos los modos, 4D Server, 4D Volume Desktop), en modo interpretado o compilado.
			***Nota**: esta opción se ofrece únicamente con fines de depuración y no debe utilizarse en producción ya que puede afectar el rendimiento de la aplicación y saturar el disco duro. Para mayor información sobre este formato y el uso del archivo 4DDebugLog[_n].txt, por favor contacte al Soporte Técnico de 4D Inc.*

Diagnostic log recording	*Entero largo*	*79*	***Alcance**: Aplicación 4D* ***Se conserva entre dos sesiones**: No* ***Valores posibles**: 0 ó 1 (0 = no guardar,1 = guardar)* ***Descripción**: inicio o detención del registro del archivo de diagnóstico de 4D. Por defecto, el valor es 0 (no guarda).* *4D permite guardar de manera continua en un archivo de diagnóstico un conjunto de eventos relativos al funcionamiento interno de la aplicación. La información contenida en este archivo está destinada a la actualización de las aplicaciones 4D y puede ser analizada con ayuda de los servicios técnicos de 4D. Cuando pasa 1 en este selector, el archivo de diagnóstico, llamado NomBase.txt, se crea automáticamente (o abre) en la carpeta **Logs** de la base. Una vez el archivo alcance un tamaño de 10 MB, se cierra y se genera un nuevo archivo NomBase_N.txt, con un número secuencial N incrementado.* *Note que es posible incluir la información personalizada en este archivo con ayuda del comando **LOG EVENT**.*
Direct2D disabled	*Entero largo*	*0*	*Ver selector 69 (Direct2D Status)*

Direct2D get active status	*Entero largo*	*74*	***Nota**: sólo puede utilizar este selector con el comando **Get database parameter** y su valor no puede definirse.*
			***Descripción**: devuelve la implementación activa de Direct2D bajo Windows.*
			***Valores posibles**: 0, 1, 2, 3, 4 o 5 (ver los valores del selector 69). El valor devuelto depende de la disponibilidad de Direct2D, del hardware y de la calidad Direct2D soportado por el sistema operativo.*
			Por ejemplo, si ejecuta:
			`SET DATABASE PARAMETER(Direct2D status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D get active status)`
			- En Windows 7 y superiores, $mode vale 1 cuando el sistema detecta un hardware compatible con Direct2D; de lo contrario, $mode valdrá 3 (contexto software).
			- En Windows Vista, $mode valdrá 1 si el sistema detecta un hardware compatible con Direct2D; de lo contrario, $mode toma el valor 0 (desactivando Direct2D).
			- En Windows XP, $mode siempre valdrá 0 (no compatible con Direct2D).

Direct2D hardware	*Entero largo*	*1*	*Ver selector 69 (Direct2D Status)*
Direct2D software	*Entero largo*	*3*	*Ver selector 69 (Direct2D Status)*

Constante	Tipo	Valor	Comentario
Direct2D status	*Entero largo*	*69*	***Alcance**: aplicación 4D* ***Se conserva entre dos sesiones**: No* ***Descripción**: modo de activación de Direct2D bajo Windows.* ***Valores posibles:** una de las siguientes constantes (modo 3 por defecto):* *Direct2D Disabled (0): el modo Direct2D no está activo, la base funciona en el modo anterior (GDI/GDIPlus).* *Direct2D Hardware (1): uso de Direct2D en contexto de hardware gráfico en toda la aplicación 4D. Si este contexto no está disponible, uso del contexto de software gráfico Direct2D (excepto bajo Vista, en cuyo caso el modo GDI/GDIPlus se utiliza para un mejor rendimiento).* *Direct2D Software (3) (Modo por defecto): a partir de Windows 7, uso de Direct2D en contexto de software gráfico en toda la aplicación 4D. En Vista, por razones de rendimiento se utiliza el modo GDI/GDIPlus.* ***Nota de compatibilidad:** a partir de 4D v14, los modos híbridos se desactivan y redireccionan a los modos disponibles (el antiguo modo 2 es equivalente a 1; los antiguos modos 4 y 5 son equivalentes al modo 3).*
HTTP compression level	*Entero largo*	*50*	***Alcance**: aplicación 4D* ***Se conserva entre dos sesiones**: no* ***Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.*
HTTP compression threshold	*Entero largo*	*51*	***Alcance**: aplicación 4D* ***Se conserva entre dos sesiones**: no* ***Valores posibles**: todo valor de tipo entero largo* ***Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.*
HTTPS Port ID	*Entero largo*	*39*	***Alcance**: 4D local, 4D Server* ***Se conserva entre dos sesiones**: sí* ***Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.*
Idle connections timeout	*Entero largo*	*54*	***Alcance**: aplicación 4D a menos que valor sea negativo* ***Se conserva entre dos sesiones:** no* ***Valores posibles:** valor entero que expresa una duración en segundos. El valor puede ser positivo (nuevas conexiones) o negativo (conexiones existentes). Por defecto, el valor es 20.* ***Descripción**: máximo periodo de inactividad (timeout) para conexiones al motor de la base 4D y al motor SQL, así como también en modo ServerNet (nueva capa de red), al servidor de la aplicación 4D. Cuando una conexión inactiva alcanza este límite, se pone en espera automáticamente, lo cual congela la sesión cliente/servidor y cierra el socket de red. En la ventana de administración del servidor, el estado del proceso del usuario se indica como "Postponed". Este funcionamiento es totalmente transparente para el usuario: tan pronto como hay una nueva actividad en la conexión que está en espera, el socket se reabre automáticamente y la sesión cliente/servidor se restaura.* *Este parámetro permite, por una parte, economizar los recursos en el servidor: las conexiones en espera cierran el socket y liberan un proceso en el servidor. Por otra parte, esto le permite evitar pérdida de conexiones por el cierre de sockets por parte del firewall. Por esta razón, el valor del timeout para conexiones inactivas deber ser menor que el del firewall en este caso.* *Si pasa un valor positivo en valor, se aplicará a todas las nuevas conexiones en todos los procesos. Si pasa un valor negativo, se aplicará a las conexiones que se abran en el proceso actual. Si pasa 0, las conexiones inactivas no serán sometidas a un timeout.* *Este parámetro puede definirse del lado del servidor y del cliente. Si pasa dos duraciones diferentes, la más corta se tendrá en cuenta. Por lo general, no necesita cambiar este valor.*
Invert objects	*Entero largo*	*37*	***Alcance**: base de datos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: 0, 1 ó 2 (0 = modo desactivado, 1 = modo automático, 2 = modo activo).* ***Descripción**: configuración del modo "inversión de los objetos" que permite invertir en modo Aplicación formularios, objetos, barras de menú, etc. cuando la base se muestra en Windows en un idioma de derecha a izquierda. Este modo también puede configurarse en la página Interfaz/Lenguajes de derecha a izquierda de las Propiedades de la base.* • *El valor 0 indica que el modo nunca ha sido activado, cualquiera que sea la configuración del sistema (corresponde al valor Nunca en las Propiedades de la base).* • *El valor 1 indica que el modo está activo o no en función de la configuración del sistema (corresponde al valor Automático en las Propiedades de la base).* • *El valor 2 indica que el modo está activo, cualquiera que sea la configuración del sistema (corresponde al valor Siempre en las Propiedades de la base).* *Para mayor información, consulte el manual de Diseño de 4D.*
Log command list	*Cadena*	*80*	***Alcance**: Aplicación 4D* ***Se conserva entre dos sesiones**: No* ***Valores posibles**: cadena que contiene la lista de números de los comandos 4D a guardar (separados por dos puntos), "all" para guardar todos los comandos o "" (cadena vacía) para no guardar ninguno.* ***Descripción**: la lista de comandos 4D a guardar en el archivo de depuración (ver el selector 34, Debug Log Recording). Por defecto, se guardan todos los comandos 4D.* *Este selector permite guardar la cantidad de información almacenada en el archivo de depuración limitando los comandos 4D donde quiera guardar la ejecución.*

Constante	Tipo	Valor	Comentario
Max concurrent Web processes	Entero largo	18	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores**: todo valor entre 10 y 32 000. El valor por defecto es 100. **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Maximum temporary memory size	Entero largo	61	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: entero largo positivo. **Descripción**: tamaño máximo de memoria temporal que 4D asignar a cada proceso, expresado en MB. Por defecto, el valor es 0 (sin tamaño máximo). 4D utiliza una memoria temporal especial dedicada a las operaciones de indexación y ordenación. Esta memoria conserva la memoria caché "estándar" durante operaciones masivas. Sólo se activa cuando es necesario. Por defecto, el tamaño de la memoria temporal está limitado únicamente por los recursos disponibles (en función de la configuración de memoria del sistema). Este mecanismo es conveniente para la mayoría de las aplicaciones. Sin embargo, en algunos contextos específicos, particularmente cuando una aplicación cliente-servidor efectúa simultáneamente un gran número de ordenaciones secuenciales, el tamaño de la memoria temporal puede aumentar críticamente, hasta volver el sistema inestable. En este contexto, fijar un tamaño máximo para la memoria temporal permite preservar el funcionamiento apropiado de la aplicación. En contraparte, la velocidad de ejecución podría afectarse: cuando se alcanza el tamaño máximo para un proceso, 4D utiliza archivos de discos, que pueden volver lentos los procesos. Para necesidades específicas tales como las descritas anteriormente, un tamaño máximo de 50 MB es generalmente un buen compromiso. Sin embargo, el valor ideal se determinará en función de las especificaciones de la aplicación y será generalmente el resultado de pruebas volumétricas en tiempo real.
Maximum Web process	Entero largo	7	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 -> 32 767 **Descripción**: número máximo de procesos web a mantener en modo no contextual con 4D en modo local y 4D Server. Por defecto, el valor es 10. En modo no contextual, para que el servidor web sea reactivo, 4D demora los procesos web 5 segundos y los reutiliza para ejecutar las posibles futuras peticiones HTTP. En términos de rendimiento, este principio es más ventajoso que crear un nuevo proceso para cada petición. Una vez se reutiliza un proceso web, se retrasa nuevamente 5 segundos. Cuando se alcanza el número máximo de procesos web, el proceso web se aborta. Si no se ha atribuido ninguna petición a un proceso web durante 5 segundos, el proceso se aborta, excepto si el número mínimo de procesos web se ha alcanzado (en cuyo caso los procesos se retrasan nuevamente). Estos parámetros le permiten ajustar el funcionamiento de su servidor web en función del número de peticiones y de la memoria disponible, como también de otros parámetros.
Maximum Web requests size	Entero largo	27	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores posibles**: 500 000 a 2 147 483 648. **Descripción**: Constante obsoleta (se conserva por compatibilidad únicamente). No se recomienda utilizar los comandos **WEB SET OPTION** y **WEB GET OPTION** para la configuración del servidor HTTP.
Min TLS version	Entero largo	105	**Alcance**: 4D Server, 4D Web Server y 4D SQL Server **Conservar entre dos sesiones**: No **Descripción**: se utiliza para especificar el nivel TLS (Transport Layer Security), que ofrece cifrado y autenticación de datos entre aplicaciones y servidores. Los valores definidos se aplican globalmente a la capa de red. Una vez modificado, el servidor debe reiniciarse para utilizar el nuevo valor. El protocolo mínimo predeterminado es TLSv1_2. Se rechazarán los intentos de conexión de clientes con un valor inferior al mínimo. **Valores posibles**: • TLSv1_0 - TLS 1.0, introducido en 1999 como una actualización de SSL v3.0. TLS 1.0 y SSL 3.0 no interoperan. • TLSv1_1 - TLS 1.1, introducido en 2006 como una actualización de TLS 1.0. Las mejoras incluyen mejor seguridad y manejo de errores, etc. • TLSv1_2 - TLS 1.2, introducido en 2008 como una actualización de TLS 1.1. Las mejoras incluyen una mayor flexibilidad, soporte adicional para el cifrado autenticado, etc. **NOTAS**: - El plugin 4D Internet Commands utiliza una capa de red diferente, por lo que este selector no tendrá ningún impacto en su versión TLS. - Se ignorarán los intentos de aplicar TLS a la capa de red heredada.
Minimum Web process	Entero largo	6	**Alcance**: 4D local, 4D Server **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 -> 32 767 **Descripción**: número mínimo de proceso web a mantener en modo no contextual con 4D en modo local y 4D Server. Por defecto, el valor es 0 (ver a continuación).

Constante	Tipo	Valor	Comentario
Number of formulas in cache	*Entero largo*	*92*	***Alcance**: aplicación 4D* ***Se conserva entre dos sesiones:** no* ***Valores posibles**: enteros largos positivos* ***Valor por defecto**: 0 (sin caché)* ***Descripción**: establece u obtiene el número máximo de fórmulas a conservar en la memoria caché de fórmulas, que es utilizado por el comando **EXECUTE FORMULA**. Este límite se aplica a todos los procesos, pero cada proceso tiene su propia caché de fórmulas. Ubicar las fórmulas en la caché acelera la ejecución del comando **EXECUTE FORMULA** en modo compilado, ya que cada fórmula en caché se tokeniza sólo una vez en este caso.Cuando se cambia el valor de la memoria caché, el contenido existente se restablecen incluso si el nuevo tamaño es más grande que el anterior. Una vez se alcanza el número máximo de fórmulas en la memoria caché, una nueva fórmula ejecutada borrará a la más antigua de la memoria caché (modo FIFO). Este parámetro sólo se tiene en cuenta en las bases o componentes compilados.*
Order by formula on server	*Entero largo*	*47*	***Alcance**: tabla y procesos actuales* ***Se conserva entre dos sesiones**: no* ***Valores posibles**: 0 (utilizar la configuración de la base), 1 (ejecutar en el cliente) o 2 (ejecutar en el servidor)* ***Descripción**: ubicación de la ejecución del comando **ORDER BY FORMULA** para la tabla pasada en parámetro.* *Al utilizar una base en modo cliente-servidor, el comando **ORDER BY FORMULA** puede ejecutarse bien sea en el equipo servidor o en el cliente. Este selector puede utilizarse para especificar la ubicación de la ejecución de este comando (servidor o cliente). Este modo también puede definirse en las preferencias de la base. Para mayor información, consulte la descripción del selector 46, Query By Formula On Server.* ***Nota:** si quiere activar las uniones "tipo SQL" (consulte el selector QUERY BY FORMULA Joins selector), siempre debe ejecutar las fórmulas en el servidor de manera que tengan acceso a los registros. Atención, en este contexto, la fórmula no debe contener llamadas a un método, de lo contrario pasará automáticamente al equipo remoto.*
PHP interpreter IP address	*Entero largo*	*55*	***Alcance**: Aplicación 4D* ***Se conserva entre dos sesiones**: No* ***Valores**: cadena formateada del tipo "nnn.nnn.nnn.nnn" (por ejemplo "127.0.0.1").* ***Descripción**: dirección IP utilizada localmente por 4D para comunicarse con el intérprete PHP vía FastCGI. Por defecto, el valor es "127.0.0.1". Esta dirección debe corresponder a la máquina donde en encuentra 4D. Este parámetro también puede definirse globalmente para todas las máquinas vía las Propiedades de la base.* *Para mayor información sobre el intérprete PHP, por favor consulte el manual de Diseño.*
PHP interpreter port	*Entero largo*	*56*	***Alcance**: Aplicación 4D* ***Se conserva entre dos sesiones**: No* ***Valores**: valor de tipo entero largo positivo. Por defecto, el valor es 8002.* ***Descripción**: número de puerto TCP utilizado o por el intérprete PHP de 4D. Este parámetro también puede modificarse globalmente para todos los equipos vía las Propiedades de la base. Para mayor información sobre el intérprete PHP, consulte el manual de Diseño.*
PHP max requests	*Entero largo*	*58*	***Alcance**: Aplicación 4D* ***Se conserva entre dos sesiones**: No* ***Valores**: valor de tipo entero largo positivo. Por defecto, el valor es 500.* ***Descripción**: número máximo de peticiones aceptadas por el intérprete PHP. Cuando se alcanza este número máximo, el intérprete devuelve errores del tipo "servidor ocupado". Por razones de seguridad o rendimiento, puede modificar este valor. Este parámetro también puede modificarse globalmente para todos los equipos vía las Propiedades de la base. Para mayor información sobre este parámetro, consulte la documentación FastCGI-PHP.* ***Nota:** de parte de 4D, estos parámetros se aplican dinámicamente; no es necesario salir de 4D para que sean tenidos en cuenta. Por otra parte, si el intérprete PHP ya fue lanzado, será necesario salir y lanzarlo nuevamente, para que las modificaciones se tengan en cuenta.*
PHP number of children	*Entero largo*	*57*	***Alcance**: Aplicación 4D* ***Se conserva entre dos sesiones**: No* ***Valores**: valor de tipo entero largo positivo. Por defecto, el valor es 5.* ***Descripción**: número de procesos hijos a crear y mantener localmente por el intérprete PHP de 4D. Por razones de optimización, el intérprete PHP crea y utiliza un conjunto (pool) de procesos sistema llamados "procesos hijos" para procesar las peticiones de ejecución de scripts. Puede variar el número de procesos hijo de acuerdo a las necesidades de su aplicación. Este parámetro también puede modificarse globalmente para todos los equipos vía las Propiedades de la base. Para mayor información sobre el intérprete PHP, consulte el manual de Diseño.* ***Nota:** bajo Mac OS, todos los procesos hijos comparten el mismo puerto. Bajo Windows, cada proceso hijo utiliza un número de puerto específico. El primer número es el definido por el intérprete PHP; los otros procesos hijos lo incrementan. Por ejemplo, si el puerto por defecto es 8002 y usted lanza 5 procesos hijos, utilizarán los puertos 8002 a 8006.*
PHP use external interpreter	*Entero largo*	*60*	***Alcance**: aplicación 4D* ***Se conserva entre dos sesiones**: no* ***Valores** : 0 = utilizar intérprete interno, 1 = utilizar intérprete externo* ***Descripción**: valor que indica si las peticiones PHP de 4D se envían al intérprete interno ofrecido por 4D o a un intérprete externo. Por defecto el valor es 0 (uso del intérprete ofrecido por 4D). Si quiere utilizar su propio intérprete PHP, por ejemplo para beneficiarse de módulos adicionales o de una configuración específica, pase 1 en valor. En este caso, 4D no lanza su intérprete interno en caso de peticiones PHP.* *El intérprete PHP personalizado debe haber sido compilado en FastCGI y estar ubicado en la misma máquina que el motor 4D. Note que en este caso, debe administrar completamente el intérprete; no será iniciado ni detenido por 4D. Este parámetro también puede modificarse globalmente para todas las máquinas vía las Propiedades de la base.*

Constante	Tipo	Valor	Comentario
Port ID	Entero largo	15	***Alcance**: 4D local, 4D Server* ***Se conserva entre dos sesiones**: no* ***Descripción**: Número de puerto TCP utilizado por el servidor web 4D con 4D en modo local y 4D Server. Por defecto, el valor es 80.* *El número de puerto TCP está definido en la página "Web/Configuración" de la caja de diálogo de las Propiedades de la base. Puede utilizar las constantes del tema para el parámetro valor.* *El selector Port ID se utiliza en el marco de servidores web 4D compilados y fusionados con 4D Desktop (sin acceso al modo Diseño). Para mayor información sobre el número de puerto TCP, consulte la sección **Parámetros del servidor web***
Query by formula joins	Entero largo	49	***Alcance**: Proceso actual* ***Se conserva entre dos sesiones**: no* ***Valores posibles**: 0 (utilizar configuración de la base), 1 (siempre utilizar relaciones automáticas) o 2 (utilizar las uniones SQL si es posible).* ***Descripción**: modo de funcionamiento de los comandos **QUERY BY FORMULA** y **QUERY SELECTION BY FORMULA** relativos al uso de "uniones SQL."* *En las bases de datos creadas a partir de la versión 11.2 de 4D v11 SQL, estos comandos efectúan uniones basados en el modelo de uniones SQL. Este mecanismo permite modificar la selección de una tabla en función de una búsqueda efectuada en otra tabla sin que las tablas estén conectadas por una relación automática (condición necesaria en las versiones anteriores de 4D).* *El selector QUERY BY FORMULA Joins permite definir el modo de funcionamiento de los comandos de búsqueda por fórmula para el proceso actual:* • *0: Utilizar los parámetros actuales de la base (valor por defecto). En bases creadas a partir de la versión 11.2 de 4D v11 SQL, las "uniones SQL" siempre se activan para las búsquedas por fórmula. En bases de datos convertidas, este mecanismo no se activa por defecto por razones de compatibilidad pero puede implementarse vía una preferencia.* • *1: Siempre utilizar relaciones automáticas (= funcionamiento de versiones anteriores de 4D). En este modo, una relación es necesaria para definir la selección de una tabla en función de búsquedas efectuadas en otra tabla. 4D no efectúa más "uniones SQL."* • *2: Utilizar las uniones SQL si es posible (= funcionamiento o defecto de las bases creadas en versión 11.2 y superiores de 4D v11 SQL). En este modo, 4D establece "uniones SQL" para las búsquedas por fórmula cuando la fórmula se ajusta para ello (con dos excepciones, ver la descripción del comando **QUERY BY FORMULA** o **QUERY SELECTION BY FORMULA**).* ***Nota:** si quiere activar las uniones "tipo SQL" (consulte el selector QUERY BY FORMULA Joins selector), siempre debe ejecutar las fórmulas en el servidor de manera que tengan acceso a los registros. Atención, en este contexto, la fórmula no debe contener llamadas a un método, de lo contrario pasará automáticamente al equipo remoto.*
Query by formula on server	Entero largo	46	***Alcance**: tabla y procesos actuales* ***Se conserva entre dos sesiones**: no* ***Valores posibles**: 0 (utilizar la configuración de la base), 1 (ejecutar en cliente) o 2 (ejecutar en servidor)* ***Descripción**: ubicación de la ejecución de los comandos **QUERY BY FORMULA** y **QUERY SELECTION BY FORMULA** para la tabla pasada en parámetro.* *Cuando se utiliza una base en modo cliente-servidor, los comandos de búsqueda "por fórmula" pueden ejecutarse en el servidor o en el equipo cliente:* • *en bases creadas con 4D v11 SQL, estos comandos se ejecutan en el servidor.* • *en bases convertidas, estos comandos se ejecutan en el equipo cliente, como en las versiones anteriores de 4D.* • *en las bases convertidas, una preferencia específica permite modificar globalmente la ubicación de ejecución de estos comandos.* *Esta diferencia en ubicación de ejecución influye no sólo en el rendimiento de la aplicación (la ejecución en el servidor es generalmente más rápida) sino también en la programación. En efecto, el valor de los componentes de la fórmula (en particular las variables llamadas vía un método) varía de acuerdo al contexto de ejecución. Puede utilizar este selector para adaptar puntualmente el funcionamiento de su aplicación.* *Si pasa 0 en el parámetro valor, la ubicación de ejecución de los comandos de búsqueda "por fórmula" dependerá de la configuración de la base: en bases creadas con 4D v11 SQL, estos comandos se ejecutarán en el servidor. En bases convertidas, se ejecutarán en el equipo cliente o en el servidor en función de las preferencias de la base. Pase 1 ó 2 en valor para "forzar" la ejecución de estos comandos respectivamente en el equipo cliente o en el servidor. Consulte el ejemplo 2.* ***Nota:** si quiere activar las uniones "tipo SQL" (consulte el selector QUERY BY FORMULA Joins selector), siempre debe ejecutar las fórmulas en el servidor de manera que tengan acceso a los registros. Atención, en este contexto, la fórmula no debe contener llamadas a un método, de lo contrario pasará automáticamente al equipo remoto.*
QuickTime support	Entero largo	82	***Alcance**: Aplicación 4D* ***Se conserva entre dos sesiones**: Sí* ***Valores posibles**: 0 (por defecto) = QuickTime desactivado, 1 = QuickTime activado.* ***Descripción**:en 4D a partir de la v14, por defecto los codecs QuickTime ya no se soportan. Por compatibilidad, puede utilizar este selector para reactivarlos en su base. La modificación de esta opción requiere que la base se reinicie. Sin embargo, debe notar que en futuras versiones de 4D, se eliminará de forma permanente el soporte QuickTime.*

Constante	Tipo	Valor	Comentario
Server base process stack size	Entero largo	53	***Alcance**: 4D Server* ***Se conserva entre dos sesiones**: no* ***Valores posibles**: entero largo positivo.* ***Descripción**: tamaño de la pila asignada a cada proceso del sistema preferente en el servidor, expresado en bytes. El tamaño por defecto es determinado por el sistema.* *Los procesos sistema preferente (procesos de tipo Proceso base 4D client) se cargan para controlar los procesos cliente 4D principales. El tamaño asignado por defecto a la pila de cada proceso preferente da facilidad de ejecución pero puede resultar consecuente cuando se crea un gran número de procesos (varios cientos).* *Por razones de optimización, este tamaño puede reducirse considerablemente si las operaciones efectuadas por la base lo permiten (por ejemplo si la base no efectúa ordenaciones de grandes cantidades de registros). Son posibles valores de 512 o incluso 256 KB. Sea cuidadoso, subdimensionar la pila es critico y puede afectar la operación de 4D Server. La definición de este parámetro debe hacerse con precaución y tener en cuenta las condiciones de uso de la base (número de registros, tipo de operaciones, etc.).* *Para que sea tenido en cuenta, este parámetro debe ejecutarse en el equipo servidor (por ejemplo en el **Método base On Server Startup**).*
Spellchecker	Entero largo	81	***Alcance**: Aplicación 4D* ***Se conserva entre dos sesiones**: No* ***Valores posibles**: 0 (por defecto) = corrector macOS nativo (Hunspell desactivado), 1 = corrector Hunspell activo.* ***Descripción**: permite activar el corrector ortográfico Hunspell bajo macOS. Por defecto, en esta plataforma el corrector nativo está activo. Puede preferir utilizar el corrector Hunspell, por ejemplo, para unificar la interfaz de sus aplicaciones multiplataformas (bajo Windows, sólo el corrector Hunspell está disponible). Para mayor información, consulte **Corrección ortográfica**.*
SQL Autocommit	Entero largo	43	***Alcance**: base de datos* ***Se conserva entre dos sesiones**: sí* ***Posibles valores**: 0 (desactivación) o 1 (activación)* ***Descripción**: activación o desactivación del modo SQL auto-commit. Por defecto, el valor es 0 (modo desactivado)* *El modo auto-commit permite reforzar la integridad referencial de la base. Cuando este modo está activo, las peticiones SELECT, **INSERT**, **UPDATE** y DELETE (SIUD) se incluyen automáticamente en las transacciones cuando no se han ejecutado dentro de una transacción. Este modo igualmente puede definirse en las Preferencias de la base.*
SQL Engine case sensitivity	Entero largo	44	***Alcance**: base de datos* ***Se conserva entre dos sesiones**: sí* ***Valores posibles**: 0 (no se tienen en cuenta las mayúsculas y minúsculas) ó 1 (sensible a las mayúsculas y minúsculas)* ***Descripción**: activación o desactivación de la sensibilidad a mayúsculas y minúsculas para comparaciones de cadenas efectuadas por el motor SQL.* *Por defecto, el valor es 1 (sensible a las mayúsculas y minúsculas): el motor SQL diferencia entre mayúsculas y minúsculas y entre caracteres acentuados al comparar cadenas (ordenaciones y búsquedas). Por ejemplo "ABC"= "ABC" pero "ABC" # "Abc." En algunos casos, por ejemplo para alinear el funcionamiento del motor SQL con el del motor 4D, podría querer que las comparaciones de cadenas no tengan en cuenta las mayúsculas y minúsculas ("ABC"="Abc").* *Esta opción también puede definirse en la **Página SQL** de las Preferencias de la base.*
SQL Server Port ID	Entero largo	88	***Alcance**: 4D modo local y 4D Server.* ***Se conserva entre dos sesiones**: Sí* ***Descripción**: permite leer o definir el número del puerto TCP utilizado por el servidor SQL integrado de 4D en modo local o 4D Server. Por defecto, el valor es 19812. Cuando se define este selector, la configuración de la base se actualiza. También puede definir el número del puerto TCP en la página "SQL" de la caja de diálogo de Propiedades de la base.* ***Valores posibles:** 0 a 65535.* ***Valor por defecto:** 19812*
SSL cipher list	Cadena	64	***Alcance**: Aplicación 4D* ***Se conserva entre dos sesiones**: No* ***Valores posibles**: secuencia de cadenas separadas por dos puntos (por ejemplo "RC4-MD5:RC4-64-MD5:....")* ***Descripción:** lista de cifrado (cipher list) utilizada por 4D para el protocolo seguro. Esta lista modifica la prioridad de los algoritmos de cifrado implementados por 4D. Por ejemplo, puede pasar la siguiente cadena en el parámetro valor:* *"AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". Para una descripción completa de la sintaxis para la lista cifrada, consulte la página de cifrado del sitio OpenSSL.* *Este parámetro es global para la aplicación (concierne al servidor HTTP, al servidor SQL, conexiones cliente/servidor, y también al cliente HTTP y a todos los comandos 4D que usan el protocolo seguro) pero es temporal (no se mantiene entre sesiones).* *Cuando la lista de cifrado se modifica, debe reiniciar el servidor correspondiente para que los nuevos parámetros sean tenidos en cuenta.* *Para reinicializar la lista de cifrado a su valor por defecto (guardado permanentemente en el archivo SLI), llame al comando **SET DATABASE PARAMETER** y pase una cadena vacía ("") en el parámetro valor.* *Por defecto, 4D utiliza el algoritmo de cifrado RC4. Si quiere utilizar el algoritmo AES (más reciente), pase la cadena siguiente en el parámetro valor:* *"AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH".* ***Nota:** con el comando **Get database parameter**, la lista de cifrado se devuelve en el parámetro opcional valorAlfa y el parámetro de retorno es siempre 0.*
String type with time zone	Entero largo	1	*Tipo de cadena con selector de zona horaria para Dates inside objects*

Constante	Tipo	Valor	Comentario
String type without time zone	Entero largo	0	Tipo de cadena sin selector de zona horaria para Dates inside objects
Table sequence number	Entero largo	31	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: sí **Valores posibles**: todo valor de tipo entero largo. **Descripción**: este selector se utiliza para modificar o modificar u obtener el número único actual de los registros de la tabla pasada en parámetro. "Número actual" significa "último número utilizado": si modifica este valor utilizando **SET DATABASE PARAMETER**, el siguiente registro será el valor pasado + 1. Este nuevo número es el número devuelto por el comando **Sequence number** como también en todo campo de la tabla a la cual se asigna la propiedad "Autoincrementar" en el editor de estructura o vía SQL. Por defecto, este número único es definido por 4D y corresponde al orden de creación de los registros. Para información adicional, por favor consulte la documentación del comando **Sequence number**.
Times in milliseconds	Entero largo	1	Selector de modo almacenamiento de horas para Times inside objects
Times in seconds	Entero largo	0	Selector de modo almacenamiento de horas para Times inside objects
Times inside objects	Entero largo	109	**Alcance**: 4D local, 4D Server (todos los procesos) **Se conserva entre dos sesiones**: No **Valores posibles**: Times in seconds (0) (predeterminado), Times in milliseconds (1) **Descripción**: define la forma en que los valores de tipo hora se convierten y almacenan dentro de las propiedades de los objetos y los elementos de la colección, así como la forma en que se importan/exportan en JSON y en las áreas web. Por defecto, a partir de 4D v17, las horas se convierten y almacenan en número de segundos en los objetos. En versiones anteriores, los valores de tiempo se convertían y almacenaban como cantidad de milisegundos en esos contextos. Usar este selector puede ayudarlo a migrar sus aplicaciones volviendo a la configuración anterior si es necesario. **Nota**: los métodos ORDA y el motor SQL ignoran esta configuración, siempre suponen que los valores de tiempo son números de segundos.
Tips delay	Entero largo	102	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: entero largo >= 0 (tics) **Descripción**: retraso antes de que se muestren las sugerencias una vez que el cursor del ratón se haya detenido en objetos con mensajes de ayuda adjuntos. El valor se expresa en tics (1/60 de segundo). El valor predeterminado es 45 tics (0.75 segundos).
Tips duration	Entero largo	103	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: entero largo >= 60 (tics) **Descripción**: duración máxima de visualización de una sugerencia. El valor se expresa en tics (1/60 de segundo). El valor predeterminado es 720 tics (12 segundos).
Tips enabled	Entero largo	101	**Alcance**: aplicación 4D **Se conserva entre dos sesiones**: No **Valores posibles**: 0 = consejos desactivados, 1 = consejos activados (predeterminado) **Descripción**: define u obtiene el estado de visualización actual de los consejos para la aplicación 4D. De forma predeterminada, las sugerencias están activadas. Tenga en cuenta que este parámetro define todos los consejos 4D, es decir, los mensajes de ayuda de formulario y las sugerencias del editor de modo Diseño.
TLSv1_0	Entero largo	1	Ver el selector 105 (Mínima versión TLS).
TLSv1_1	Entero largo	2	Ver el selector 105 (Mínima versión TLS).
TLSv1_2	Entero largo	3	Ver el selector 105 (Mínima versión TLS).
Unicode mode	Entero largo	41	**Alcance**: base de datos **Se conserva entre dos sesiones**: sí **Valores posibles**: 0 (modo compatibilidad) ó 1 (modo Unicode) **Descripción**: modo de ejecución actual de la base, relativo al conjunto de caracteres. 4D soporta el conjunto de caracteres Unicode pero puede funcionar en modo "compatibilidad" (basado en el conjunto de caracteres Mac ASCII). Por defecto, las bases de datos convertidas se ejecutan en modo compatibilidad (0) y las bases creadas a partir de la versión 11 o superior se ejecutan en modo Unicode. El modo de ejecución puede controlarse vía una opción de las Preferencias y también puede leerse o (con propósitos de realizar pruebas) modificarse vía este selector. La modificación de esta opción necesita que la base se reinicie para que sea tenida en cuenta. Note que dentro de un componente no es posible modificar este valor, sólo leerlo.

Constante	Tipo	Valor	Comentario
Use legacy network layer	*Entero largo*	*87*	***Alcance:*** *4D en modo local, 4D Server* ***Se conserva entre dos sesiones:*** *sí* ***Descripción:*** *fija u obtiene el estado actual de la capa de red antigua para las conexiones cliente/servidor.* *La capa de red antigua es obsoleta a partir de 4D v14 R5 y debe ser reemplazada progresivamente en sus aplicaciones por la capa de red ServerNet. ServerNet será requerida en próximas versiones 4D con el fin de beneficiarse de las futuras evoluciones de la red. Por razones de compatibilidad, la capa de red antigua aún se soporta para permitir una transición sin problemas para las aplicaciones existentes; (se usa por defecto en aplicaciones convertidas de una versión anterior a v14 R5). Pase 1 en este parámetro para utilizar la capa de red antigua (y desactivar ServerNet) para las conexiones cliente/servidor, y pase 0 para deshabilitar la red antigua (y utilizar ServerNet).* *Esta propiedad también se puede definir mediante la opción "Usar capa de red antigua " que se encuentran en* ***Página Compatibilidad*** *de las Propiedades de la base (ver* ***Opciones red y cliente-servidor****). En esta sección, también puede encontrar una discusión sobre la estrategia de migración. Le recomendamos que active ServerNet tan pronto como sea posible.* *Deberá reiniciar la aplicación para que este parámetro sea tenido en cuenta. No está disponible en 4D Server v14 R5 64-bit versión para OS X, que sólo soporta el ServetNet; (siempre devuelve 0).* ***Valores posibles:*** *0 o 1 (0 = no utilizan capa de red antigua, 1 = uso capa de red antigua)* ***Valor por defecto:*** *0 en bases de datos creadas con 4D v14 R5 o superior, 1 en bases de datos convertidas de 4D v14 R4 o anteriores.*

Parámetros Gráficos

Constante	Tipo	Valor	Comentario
Graph background color	Cadena	graphBackgroundColor	**Valores posibles**: expresión de color compatible con SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414"
Graph background opacity	Cadena	graphBackgroundOpacity	**Valores posibles**:enteros, rango de 0-100 **Valor por defecto**: 100
Graph background shadow color	Cadena	graphBackgroundShadowColor	**Valores posibles**: expresión de color compatible con SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414"
Graph bottom margin	Cadena	bottomMargin	**Valores posibles**: números reales **Valor por defecto**: 12
Graph colors	Cadena	colors	**Valores posibles**: array texto. Los colores para cada serie de gráfico. **Valores por defecto**: Blue-green (#19BAC9), Yellow (#FFC338), Purple (#573E82), Green (#4FA839), Orange (#D95700), Blue (#1D9DF2), Yellow-green (#B5CF32), Red (#D43A26)
Graph column gap	Cadena	columnGap	**Valores posibles**:enteros **Valor por defecto**: 12 Define el espacio entre las barras Tipos 1, 2, 3 únicamente
Graph column width max	Cadena	columnWidthMax	**Valores posibles**: números reales **Valor por defecto**: 200 Tipos 1, 2, 3 únicamente
Graph column width min	Cadena	columnWidthMin	**Valores posibles**: números reales **Valor por defecto**: 10 Tipos 1, 2, 3 únicamente
Graph default height	Cadena	defaultHeight	**Valores posibles**: números reales **Valor por defecto**: 400. Si graphType=7 (Pie), luego valor por defecto = 600
Graph default width	Cadena	defaultWidth	**Valores posibles**: números reales **Valor por defecto:** 600. Si graphType=7 (Pie), luego valor por defecto = 800
Graph display legend	Cadena	displayLegend	**Valores posibles:** Booleano **Valor por defecto:** True
Graph document background color	Cadena	documentBackgroundColor	**Valores posibles**: expresión color SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414". Cuando un gráfico guardado como imagen SVG se abre en otro lugar, el color de fondo del documento sólo se muestra si el motor de renderizado soporta la norma SVG Tiny 1.2 (soportado por IE, Firefox, pero no en Chrome).
Graph document background opacity	Cadena	documentBackgroundOpacity	**Valores posibles**: entero, rango 0-100 (valor por defecto: 100). Cuando un gráfico guardado como imagen SVG se abre en otro lugar, el color de fondo del documento sólo se muestra si el motor de renderizado soporta la norma SVG Tiny 1.2 (soportado por IE, Firefox, pero no en Chrome).
Graph font color	Cadena	fontColor	**Valores posibles:** expresión color SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414"
Graph font size	Cadena	fontSize	**Valores posibles**: enteros largos **Valor por defecto**: 12. Si graphType=7 (Pie), ver Graph pie font size
Graph left margin	Cadena	leftMargin	**Valores posibles**: números reales **Valor por defecto**: 12
Graph legend font color	Cadena	legendFontColor	**Valores posibles**: expresión de color compatible con SVG (texto), por ejemplo "#7F8E00", "Pink", o "#0a1414"
Graph legend icon gap	Cadena	legendIconGap	**Valores posibles**: números reales **Valor por defecto**: Graph legend icon height/2
Graph legend icon height	Cadena	legendIconHeight	**Valores posibles**: números reales **Valor por defecto**: 20
Graph legend icon width	Cadena	legendIconWidth	**Valores posibles**: números reales **Valor por defecto**: 20
Graph legend labels	Cadena	legendLabels	**Valores posibles:** array texto. Si falta, 4D muestra íconos sin texto.
Graph line width	Cadena	lineWidth	**Valores posibles**: números reales **Valor por defecto**: 2 Tipo 4 únicamente
Graph pie direction	Cadena	pieDirection	**Valores posibles**: 1 o -1 **Valor por defecto**: 1 1 indica la dirección de las agujas del reloj, -1 indica la dirección en sentido antihorario

Constante	Tipo	Valor	Comentario
Graph pie font size	Cadena	pieFontSize	**Valores posibles**: números reales **Valor por defecto**: 16 Tipo 7 únicamente
Graph pie shift	Cadena	pieShift	**Valores posibles**: números reales **Valor por defecto**: 8 Tipo 7 únicamente
Graph pie start angle	Cadena	pieStartAngle	**Valores posibles**: números reales (positivos o negativos) **Valor por defecto**: 0, que representa un ángulo de inicio de 0° (posición hacia arriba) Un valor positivo representa un ángulo relativo a la dirección actual del pie. Un valor negativo representa un ángulo relativo a la dirección opuesta al pie
Graph plot height	Cadena	plotHeight	**Valores posibles**: números reales **Valor por defecto**: 12 Tipo 4 únicamente
Graph plot radius	Cadena	plotRadius	**Valores posibles**: números reales **Valor por defecto**: 12 Tipo 6 únicamente
Graph plot width	Cadena	plotWidth	**Valores posibles**: números reales **Valor por defecto**: 12 Tipo 4 únicamente
Graph right margin	Cadena	rightMargin	**Valores posibles**: números reales **Valor por defecto**: 2
Graph top margin	Cadena	topMargin	**Valores posibles**: números reales **Valor por defecto**: 2
Graph type	Cadena	graphType	**Valores posibles**: enteros largos [1 a 8], donde 1 = barras, 2 = proporcional, 3 = apilados, 4 = líneas, 5 = superficies, 6 = puntos, 7 = pie, 8 = imágenes. **Valor por defecto**: 1 Si es nulo, el gráfico no se dibuja y no se muestra ningún mensaje de error. Si está fuera de rango, el gráfico no se dibuja y se muestra un mensaje de error. Si desea modificar gráficos de tipo imagen (valor=8), debe copiar la carpeta 4D/Resources/GraphTemplates/Graph_8_Pictures/ en la carpeta Resources de su base y realizar las modificaciones necesarias. Los archivos imagen locales se utilizarán en lugar de los archivos 4D. No hay un patrón de nombres de imágenes; 4D ordena los archivos contenidos en la carpeta y le asigna el primer archivo al primer gráfico. Estos archivos pueden ser del tipo SVG o imagen.
Graph xGrid	Cadena	xGrid	**Valores posibles**: Booleano **Valor por defecto**: True Todos los tipos excepto 7
Graph xMax	Cadena	xMax	**Valores posibles**: número, fecha, hora (mismo tipo que el parámetro xLabels). Sólo los valores más altos que xMax se muestran en el gráfico. xMax se utiliza solamente para los tipos de gráficos 4, 5 o 6 si xProp = true y si el tipo xLabels es un número, fecha u hora. Si falta o si xMin>xMax, 4D calcula automáticamente el valor xMax.
Graph xMin	Cadena	xMin	**Valores posibles**: número, fecha, hora (mismo tipo que el parámetro xLabels). Sólo los valores más altos que xMin se muestran en el gráfico. xMin se utiliza solamente para los tipos de gráficos 4, 5 o 6 si xProp = true y si el tipo xLabels es un número, fecha u hora. Si falta o si xMin>xMax, 4D calcula automáticamente el valor xMin.
Graph xProp	Cadena	xProp	**Valores posibles**: Booleano **Valor por defecto:**True True para eje x proporcional; False para eje x normal. xProp se utiliza únicamente para los tipos de gráficos 4, 5 o 6
Graph yGrid	Cadena	yGrid	**Valores posibles**: Booleano **Valor por defecto**: True Todos los tipos excepto 7
Graph yMax	Cadena	yMax	**Valores posibles**: números Si falta, 4D calcula automáticamente el valor yMax. Todos los tipos excepto 7
Graph yMin	Cadena	yMin	**Valores posibles**: números Si falta, 4D calcula automáticamente el valor yMin. Todos los tipos excepto 7

PHP

Constante	Tipo	Valor	Comentario
PHP privileges	Entero largo	1	Definition of specific user privileges relating to the execution of the script. **Possible value(s)**: String in the form "User:Password". For example: "root:jd51254d"
PHP raw result	Entero largo	2	Definition of processing mode for HTTP headers returned by PHP in the execution result when this result is of the Text type (when the result is of the BLOB type, headers are always kept). **Possible value(s)**: Boolean. False (default value = remove HTTP headers from result. True = keep HTTP headers.

Picture Metadata Values

Constante	Tipo	Valor	Comentario
EXIF Action	Entero largo	6	
EXIF Adobe RGB	Entero largo	2	
EXIF Aperture Priority AE	Entero largo	3	
EXIF Auto	Entero largo	0	
EXIF Auto Bracket	Entero largo	2	
EXIF Auto Mode	Entero largo	3	
EXIF Average	Entero largo	1	
EXIF B	Entero largo	6	
EXIF Cb	Entero largo	2	
EXIF Center Weighted Average	Entero largo	2	
EXIF Close	Entero largo	2	
EXIF Cloudy	Entero largo	10	
EXIF Color Sequential Area	Entero largo	5	
EXIF Color Sequential Linear	Entero largo	8	
EXIF Compulsory Flash Firing	Entero largo	1	
EXIF Compulsory Flash Suppression	Entero largo	2	
EXIF Cool White Fluorescent	Entero largo	14	
EXIF Cr	Entero largo	3	
EXIF Creative	Entero largo	5	
EXIF Custom	Entero largo	1	
EXIF D50	Entero largo	23	
EXIF D55	Entero largo	20	
EXIF D65	Entero largo	21	
EXIF D75	Entero largo	22	
EXIF Day White Fluorescent	Entero largo	13	
EXIF Daylight	Entero largo	1	
EXIF Daylight Fluorescent	Entero largo	12	
EXIF Detected	Entero largo	3	
EXIF Digital Camera	Entero largo	3	
EXIF Distant	Entero largo	3	
EXIF Exposure Portrait	Entero largo	7	
EXIF Film Scanner	Entero largo	1	
EXIF Fine Weather	Entero largo	9	
EXIF Flashlight	Entero largo	4	
EXIF G	Entero largo	5	
EXIF High	Entero largo	2	
EXIF High Gain Down	Entero largo	4	
EXIF High Gain Up	Entero largo	2	
EXIF ISOStudio Tungsten	Entero largo	24	
EXIF Landscape	Entero largo	8	
EXIF Light Fluorescent	Entero largo	2	
EXIF Low	Entero largo	1	
EXIF Low Gain Down	Entero largo	3	
EXIF Low Gain Up	Entero largo	1	
EXIF Macro	Entero largo	1	
EXIF Manual	Entero largo	1	
EXIF Multi Segment	Entero largo	5	
EXIF Multi Spot	Entero largo	4	
EXIF Night	Entero largo	3	
EXIF No Detection Function	Entero largo	0	
EXIF None	Entero largo	0	
EXIF Normal	Entero largo	0	
EXIF Not Defined	Entero largo	1	
EXIF Not Detected	Entero largo	2	
EXIF One Chip Color Area	Entero largo	2	
EXIF Other	Entero largo	255	
EXIF Partial	Entero largo	6	
EXIF Program AE	Entero largo	2	
EXIF R	Entero largo	4	
EXIF Reflection Print Scanner	Entero largo	2	
EXIF Reserved	Entero largo	1	
EXIF s RGB	Entero largo	1	
EXIF Scene Landscape	Entero largo	1	
EXIF Scene Portrait	Entero largo	2	
EXIF Shade	Entero largo	11	
EXIF Shutter Speed Priority AE	Entero largo	4	
EXIF Spot	Entero largo	3	

Constante	Tipo	Valor	Comentario
EXIF Standard	Entero largo	0	
EXIF Standard Light A	Entero largo	17	
EXIF Standard Light B	Entero largo	18	
EXIF Standard Light C	Entero largo	19	
EXIF Three Chip Color Area	Entero largo	4	
EXIF Trilinear	Entero largo	7	
EXIF Tungsten	Entero largo	3	
EXIF Two Chip Color Area	Entero largo	3	
EXIF Uncalibrated	Entero largo	-1	
EXIF Unknown	Entero largo	0	
EXIF Unused	Entero largo	0	
EXIF White Fluorescent	Entero largo	15	
EXIF Y	Entero largo	1	
GPS 2D	Entero largo	2	
GPS 3D	Entero largo	3	
GPS Above Sea Level	Entero largo	0	
GPS Below Sea Level	Entero largo	1	
GPS Correction Applied	Entero largo	1	
GPS Correction Not Applied	Entero largo	0	
GPS East	Cadena	E	
GPS km h	Cadena	K	
GPS knots h	Cadena	K	
GPS Magnetic north	Cadena	M	
GPS Measurement in progress	Cadena	A	
GPS Measurement Interoperability	Cadena	V	
GPS miles h	Cadena	M	
GPS North	Cadena	N	
GPS South	Cadena	S	
GPS True north	Cadena	T	
GPS West	Cadena	W	
IPTC Action	Entero largo	11900	
IPTC Aerial View	Entero largo	11200	
IPTC Close Up	Entero largo	11800	
IPTC Couple	Entero largo	10700	
IPTC Exterior View	Entero largo	11600	
IPTC Full Length	Entero largo	10300	
IPTC General View	Entero largo	11000	
IPTC Group	Entero largo	10900	
IPTC Half Length	Entero largo	10200	
IPTC Headshot	Entero largo	10100	
IPTC Interior View	Entero largo	11700	
IPTC Movie Scene	Entero largo	12400	
IPTC Night Scene	Entero largo	11400	
IPTC Off Beat	Entero largo	12300	
IPTC Panoramic View	Entero largo	11100	
IPTC Performing	Entero largo	12000	
IPTC Posing	Entero largo	12100	
IPTC Profile	Entero largo	10400	
IPTC Rear View	Entero largo	10500	
IPTC Satellite	Entero largo	11500	
IPTC Single	Entero largo	10600	
IPTC Symbolic	Entero largo	12200	
IPTC Two	Entero largo	10800	
IPTC Under Water	Entero largo	11300	
TIFF Adobe Deflate	Entero largo	8	
TIFF Black Is Zero	Entero largo	1	
TIFF CCIRLEW	Entero largo	32771	
TIFF CCITT1D	Entero largo	2	
TIFF CIELab	Entero largo	8	
TIFF CM	Entero largo	3	
TIFF CMYK	Entero largo	5	
TIFF Color Filter Array	Entero largo	32803	
TIFF DCS	Entero largo	32947	
TIFF Deflate	Entero largo	32946	
TIFF Epson ERF	Entero largo	32769	
TIFF Horizontal	Entero largo	1	
TIFF ICCLab	Entero largo	9	

Constante	Tipo	Valor	Comentario
TIFF Inches	Entero largo	2	
TIFF IT8BL	Entero largo	32898	
TIFF IT8CTPAD	Entero largo	32895	
TIFF IT8LW	Entero largo	32896	
TIFF IT8MP	Entero largo	32897	
TIFF ITULab	Entero largo	10	
TIFF JBIG	Entero largo	34661	
TIFF JBIGB&W	Entero largo	9	
TIFF JBIGColor	Entero largo	10	
TIFF JPEG	Entero largo	7	
TIFF JPEG2000	Entero largo	34712	
TIFF JPEGThumbs Only	Entero largo	6	
TIFF Kodak DCR	Entero largo	65000	
TIFF Kodak KDC	Entero largo	32867	
TIFF Kodak262	Entero largo	262	
TIFF Linear Raw	Entero largo	34892	
TIFF LZW	Entero largo	5	
TIFF MDIBinary Level Codec	Entero largo	34718	
TIFF MDIProgressive Transform Codec	Entero largo	34719	
TIFF MDIVector	Entero largo	34720	
TIFF Mirror Horizontal	Entero largo	2	
TIFF Mirror Horizontal And Rotate270CW	Entero largo	5	
TIFF Mirror Horizontal And Rotate90CW	Entero largo	7	
TIFF Mirror Vertical	Entero largo	4	
TIFF MM	Entero largo	4	
TIFF Next	Entero largo	32766	
TIFF Nikon NEF	Entero largo	34713	
TIFF None	Entero largo	1	
TIFF Pack Bits	Entero largo	32773	
TIFF Pentax PEF	Entero largo	65535	
TIFF Pixar Film	Entero largo	32908	
TIFF Pixar Log	Entero largo	32909	
TIFF Pixar Log L	Entero largo	32844	
TIFF Pixar Log Luv	Entero largo	32845	
TIFF RGB	Entero largo	2	
TIFF RGBPalette	Entero largo	3	
TIFF Rotate180	Entero largo	3	
TIFF Rotate270CW	Entero largo	8	
TIFF Rotate90CW	Entero largo	6	
TIFF SGILog	Entero largo	34676	
TIFF SGILog24	Entero largo	34677	
TIFF Sony ARW	Entero largo	32767	
TIFF T4Group3Fax	Entero largo	3	
TIFF T6Group4Fax	Entero largo	4	
TIFF Thunderscan	Entero largo	32809	
TIFF Transparency Mask	Entero largo	4	
TIFF UM	Entero largo	5	
TIFF Uncompressed	Entero largo	1	
TIFF White Is Zero	Entero largo	0	
TIFF YCb Cr	Entero largo	6	

Portapapeles

Constante	*Tipo*	*Valor*	*Comentario*
No such data in pasteboard	*Entero largo*	*-102*	
Picture data	*Cadena*	*PICT*	
Text data	*Cadena*	*TEXT*	

Profundidad de la pantalla

Constante	*Tipo*	*Valor*	*Comentario*
Black and white	*Entero largo*	*0*	
Four colors	*Entero largo*	*2*	
Is color	*Entero largo*	*1*	
Is gray scale	*Entero largo*	*0*	
Millions of colors 24 bit	*Entero largo*	*24*	
Millions of colors 32 bit	*Entero largo*	*32*	
Sixteen colors	*Entero largo*	*4*	
Thousands of colors	*Entero largo*	*16*	
Two fifty six colors	*Entero largo*	*8*	

Propiedades de ítem de menú

Constante	*Tipo*	*Valor*	*Comentario*
Access privileges	*Cadena*	*4D_access_group*	*Activar la opción "Iniciar un nuevo proceso"* *0 = No, 1 = Sí*
Associated standard action	*Cadena*	*4D_standard_action*	*Asociar una acción estándar a la línea de menú* *Ver las constantes del tema "**Acción estándar**"*
Start a new process	*Cadena*	*4D_start_new_process*	*Asignar un grupo de acceso al comando* *0 = Sin restricción* *>0 = Número de grupo*

Propiedades de la plataforma

Constants prefixed with _o_ are obsolete and cannot be returned by the command. They are only kept for compatibility.

Constante	Tipo	Valor	Comentario
_o_INTEL 386	Entero largo	386	
_o_INTEL 486	Entero largo	486	
_o_Macintosh 68K	Entero largo	1	
_o_PowerPC 601	Entero largo	601	
_o_PowerPC 603	Entero largo	603	
_o_PowerPC 604	Entero largo	604	
_o_PowerPC G3	Entero largo	510	
Intel compatible	Entero largo	586	
Mac OS	Entero largo	2	
Power PC	Entero largo	406	
Windows	Entero largo	3	

Propiedades de los objetos

Constante	Tipo	Valor	Comentario
Align bottom	Entero largo	4	
Align center	Entero largo	3	
Align default	Entero largo	1	
Align left	Entero largo	2	
Align right	Entero largo	4	
Align top	Entero largo	2	
Asynchronous progress bar	Entero largo	3	Indicador circular muestra una animación continua
Barber shop	Entero largo	2	Barra que muestra una animación continua
Border Dotted	Entero largo	2	Los objetos aparecen enmarcados con una línea punteada de 1-pt.
Border Double	Entero largo	5	Los objetos aparecen enmarcados con una línea doble, es decir, dos líneas continuas de 1-pt. separadas por un píxel
Border None	Entero largo	0	Los objetos aparecen sin borde
Border Plain	Entero largo	1	Los objetos aparecen enmarcado con una línea de borde continua de 1-pt.
Border Raised	Entero largo	3	Los objetos aparecen con un efecto 3D (relieve)
Border Sunken	Entero largo	4	Los objetos aparecen enmarcados con un efecto 3D hundido (relieve inverso)
Border System	Entero largo	6	La línea del borde se dibuja en función de las especificaciones gráficas del sistema
Choice list	Entero largo	0	Lista simple de selección de valores (opción "Lista" en la Lista de Propiedades) (por defecto)
Disable events others unchanged	Entero largo	2	Todos los eventos listados en el array arrEvents se desactivan; el estado de todos los demás eventos no cambia
Enable events disable others	Entero largo	0	Todos los eventos listados en el array arrEvents se activan; todos los demás eventos se desactivan
Enable events others unchanged	Entero largo	1	Todos los eventos listados en el array arrEvents se activan; el estado de todos los demás eventos no cambia
Excluded list	Entero largo	2	Lista de valores no aceptados para la entrada (Opción "Exclusiones" en la lista de propiedades)
Multiline Auto	Entero largo	0	En las áreas de una sola línea, las palabras situadas al final de las líneas se cortan y no hay retornos de línea. En las áreas de varias líneas, 4D efectúa saltos de línea automáticos.
Multiline No	Entero largo	2	Nunca hay vuelta de la línea: el texto se muestra siempre en una sola línea. Si el campo o la variable alfa o texto contiene retornos de carro, el texto situado después del primer retorno de carro se elimina tan pronto como se modifica el área.
Multiline Yes	Entero largo	1	En las áreas de una sola línea, el texto se muestra hasta el primer retorno de carro o hasta la última palabra que se puede mostrar por completo. 4D inserta retornos de línea, es posible desplazarse por el contenido del área con la tecla de flecha hacia abajo. En las áreas de varias líneas, 4D efectúa los saltos de línea automáticos.
Orientation 0°	Entero largo	0	Sin rotación (valor por defecto)
Orientation 180°	Entero largo	180	Orientación del texto a 180° en el sentido horario
Orientation 90° left	Entero largo	270	Orientación del texto a 90° en el sentido antihorario
Orientation 90° right	Entero largo	90	Orientación del texto a 90° en el sentido horario
Print Frame fixed with multiple records	Entero largo	2	El tamaño inicial del marco permanece del mismo tamaño, 4D imprime el formulario varias veces para incluir todos los registros.
Print Frame fixed with truncation	Entero largo	1	4D imprime sólo los registros que aparecen en el área del subformulario. El formulario se imprime sólo una vez y los registros que no se imprimen se ignoran.
Progress bar	Entero largo	1	Barra de progreso estándar
Required list	Entero largo	1	Lista sólo los valores aceptados para la entrada (Opción "Obligatoria" en la Lista de Propiedades)
Resize horizontal grow	Entero largo	1	Si la ventana se agranda un 50% de ancho, el objeto se agranda 50% a la derecha

Constante	*Tipo*	*Valor*	*Comentario*
Resize horizontal move	*Entero largo*	*2*	*Si la ventana se agranda 100 píxeles de ancho, el objeto se mueve 100 píxeles a la derecha*
Resize horizontal none	*Entero largo*	*0*	*Si la ventana se agranda de ancho, ni el largo ni la posición del objeto varían*
Resize vertical grow	*Entero largo*	*1*	*Si la ventana se agranda un 50% de alto, el objeto se alarga 50% hacia abajo*
Resize vertical move	*Entero largo*	*2*	*Si la ventana se agranda 100 píxeles de alto, el objeto se mueve 100 píxeles hacia abajo*
Resize vertical none	*Entero largo*	*0*	*Si la ventana se agranda de alto, ni el ancho ni la posición del objeto varían*

Propiedades de los recursos

Constante	*Tipo*	*Valor*	*Comentario*
Changed resource bit	*Entero largo*	*1*	
Changed resource mask	*Entero largo*	*2*	
Locked resource bit	*Entero largo*	*4*	
Locked resource mask	*Entero largo*	*16*	
Preloaded resource bit	*Entero largo*	*2*	
Preloaded resource mask	*Entero largo*	*4*	
Protected resource bit	*Entero largo*	*3*	
Protected resource mask	*Entero largo*	*8*	
Purgeable resource bit	*Entero largo*	*5*	
Purgeable resource mask	*Entero largo*	*32*	
System heap resource bit	*Entero largo*	*6*	
System heap resource mask	*Entero largo*	*64*	

QR Bordes

Constante	Tipo	Valor	Comentario
qr bottom border	Entero largo	8	Borde inferior
qr inside horizontal border	Entero largo	32	Borde interior horizontal
qr inside vertical border	Entero largo	16	Borde interior vertical
qr left border	Entero largo	1	Borde izquierdo
qr right border	Entero largo	4	Borde derecho
qr top border	Entero largo	2	Borde superior

QR Comandos

Constante	Tipo	Valor	Comentario
qr cmd 4D View destination	Entero largo	2503	
qr cmd add column	Entero largo	2608	
qr cmd alt back color palette	Entero largo	1004	
qr cmd automatic width	Entero largo	2605	
qr cmd average	Entero largo	507	
qr cmd back color palette	Entero largo	1003	
qr cmd back colors toolbar	Entero largo	2052	
qr cmd bold	Entero largo	500	
qr cmd borders	Entero largo	2609	
qr cmd center justified	Entero largo	504	
qr cmd columns toolbar	Entero largo	2054	
qr cmd count	Entero largo	510	
qr cmd default justified	Entero largo	512	
qr cmd delete column	Entero largo	2601	
qr cmd disk file destination	Entero largo	2501	
qr cmd edit column	Entero largo	2603	
qr cmd font color palette	Entero largo	1002	
qr cmd font dropdown	Entero largo	1000	
qr cmd format	Entero largo	2606	
qr cmd generate	Entero largo	2008	Compatible editor 64 bits (uso del comando **QR RUN** recomendado)
qr cmd graph destination	Entero largo	2502	
qr cmd header and footer	Entero largo	2005	
qr cmd hide column	Entero largo	2602	
qr cmd hide line	Entero largo	2607	
qr cmd HTML file destination	Entero largo	2504	
qr cmd insert column	Entero largo	2600	
qr cmd italic	Entero largo	501	
qr cmd left justified	Entero largo	503	
qr cmd max	Entero largo	509	
qr cmd min	Entero largo	508	
qr cmd move left	Entero largo	3002	
qr cmd move right	Entero largo	3003	
qr cmd new	Entero largo	2000	
qr cmd open	Entero largo	2001	
qr cmd operators toolbar	Entero largo	2051	
qr cmd page setup	Entero largo	2006	Compatible editor 64 bits
qr cmd plain	Entero largo	511	
qr cmd presentation	Entero largo	2611	
qr cmd print preview	Entero largo	2007	Compatible editor 64 bits
qr cmd printer destination	Entero largo	2500	
qr cmd repeated values	Entero largo	2604	
qr cmd revert to save	Entero largo	2004	
qr cmd right justified	Entero largo	505	
qr cmd save	Entero largo	2002	
qr cmd save as	Entero largo	2003	
qr cmd standard deviation	Entero largo	513	
qr cmd standard toolbar	Entero largo	2053	
qr cmd style toolbar	Entero largo	2050	
qr cmd sum	Entero largo	506	
qr cmd totals spacing	Entero largo	2610	
qr cmd underline	Entero largo	502	

QR Destino de salida

Constante	Tipo	Valor	Comentario
_o_qr 4D Chart area	Entero largo	4	*** Constante obsoleta ***
qr 4D View area	Entero largo	3	detalles: N.A.
qr HTML file	Entero largo	5	detalles: ruta de acceso al archivo.
qr printer	Entero largo	1	detalles: "*" para eliminar las cajas de diálogo de impresión
qr text file	Entero largo	2	detalles: ruta de acceso al archivo.

QR Filas para propiedades

Constante	Tipo	Valor	Comentario
qr detail	Entero largo	-2	Área de detalle del informe
qr footer	Entero largo	-5	Pie de página
qr grand total	Entero largo	-3	Área total general
qr header	Entero largo	-4	Encabezado de página
qr title	Entero largo	-1	Título del informe

QR Operadores

Constante	Tipo	Valor	Comentario
qr average	Entero largo	2	
qr count	Entero largo	16	
qr max	Entero largo	8	
qr min	Entero largo	4	
qr standard deviation	Entero largo	32	
qr sum	Entero largo	1	

QR Propiedades de área

Constante	Tipo	Valor	Comentario
qr view color toolbar	Entero largo	5	Muestra la barra de herramientas Colores (Mostrada=1, Oculta=0)
qr view column toolbar	Entero largo	6	Muestra la barra de herramientas Columnas (Mostrada=1, Oculta=0)
qr view contextual menus	Entero largo	7	Muestra los menús contextuales (Mostrados=1, Ocultos=0)
qr view menubar	Entero largo	1	Muestra la barra de menús (Mostrada=1, Oculta=0)
qr view operators toolbar	Entero largo	4	Muestra la barra de herramientas Operadores (Mostrada=1, Oculta=0)
qr view standard toolbar	Entero largo	2	Muestra la barra de herramientas estándar (Mostrada=1, Oculta=0)
qr view style toolbar	Entero largo	3	Muestra la barra de herramientas Estilo (Mostrada=1, Oculta=0)

QR Propiedades de documento

Constante	Tipo	Valor	Comentario
qr printing dialog	*Entero largo*	*1*	*Visualización de la caja de diálogo de impresión:* • *Si valor = 0, la caja de diálogo de impresión no se muestra antes de la impresión.* • *Si valor = 1, la caja de diálogo de impresión se muestra antes de la impresión (valor por defecto).*
qr unit	*Entero largo*	*2*	*Unidad del documento:* • *Si valor = 0, la unidad del documento es el punto.* • *Si valor = 1, la unidad del documento es el centímetro.* • *Si valor = 2, la unidad del documento es la pulgada.*

QR Propiedades de texto

Constante	Tipo	Valor	Comentario
_o_qr font	Entero largo	1	Obsoleto desde 4D v14R3 (utilice qr font name)
qr alternate background color	Entero largo	9	Número de color de fondo alterno
qr background color	Entero largo	8	Número de color de fondo
qr bold	Entero largo	3	Atributo negrita (0 ó 1)
qr font name	Entero largo	10	Nombre de la fuente como la devuelve por ejemplo el comando **FONT LIST**
qr font size	Entero largo	2	Tamaño de fuente expresado en puntos (9 a 255)
qr italic	Entero largo	4	Atributo itálica (0 ó 1)
qr justification	Entero largo	7	Atributo de justificación (0 por defecto, 1 a la izquierda, 2 al centro y 3 a la derecha)
qr text color	Entero largo	6	Atributo de número de color (Entero largo)
qr underline	Entero largo	5	Atributo de estilo subrayado (0 ó 1)

QR Tipos de informes

Constante	Tipo	Valor	Comentario
qr cross report	Entero largo	2	
qr list report	Entero largo	1	

Relaciones

Constante	Tipo	Valor	Comentario
Automatic	*Entero largo*	*3*	
Do not modify	*Entero largo*	*0*	
Manual	*Entero largo*	*2*	
No relation	*Entero largo*	*0*	
Structure configuration	*Entero largo*	*1*	

Servicios Web (Cliente)

Constante	Tipo	Valor	Comentario
Web Service compression	Entero largo	1	
Web Service detailed message	Entero largo	1	Mensaje detallado que describe el error. El tipo de mensaje difiere según el tipo de error principal. - Si el error principal = 9910 (Error Soap): se devuelve la causa del error SOAP (ej.: "el método remoto no existe"). - Si el error principal = 9911 (Error de analizador xml): se devuelve la ubicación del error en el documento XML. - Si el error principal = 9912 (Error HTTP): - Si el error HTTP se ubica en el intervalo [300-400] (problemas relacionados con la ubicación del documento solicitado), se devuelve la nueva ubicación del URL solicitado. - Para todo otro código de error HTTP, se devuelve el <body>. - Si el error principal = 9913 (Error de red): se devuelve la causa del error de red (ej.: "ServerAddress: error DNS") - Si el error principal = 9914 (Error interno): se devuelve la causa del error interno
Web Service display auth dialog	Entero largo	4	valor = 0 (no mostrar la caja de diálogo) ó 1 (mostrar caja de diálogo) Esta opción administra la visualización de la caja de diálogo de actualización durante la ejecución del comando **CALL WEB SERVICE**. Por defecto, este comando nunca muestra la caja de diálogo; por lo general, para hacerlo debe utilizar el comando **AUTHENTICATE WEB SERVICE**. Sin embargo, si quiere que aparezca la caja de diálogo de autenticación para que el usuario introduzca sus identificadores, deberá utilizar esta opción: pase 1 en valor para mostrar la caja de diálogo, de lo contrario pase 0. La caja de diálogo sólo aparece si el servicio web necesita autenticación.
Web Service dynamic	Entero largo	0	
Web Service error code	Entero largo	0	Código del error principal (definido por 4D). Este código también es devuelto en la variable sistema Error. Lista de códigos que pueden ser devueltos: 9910: Error Soap (ver también Web Service Fault Actor) 9911: Error de analizador xml 9912: Error HTTP (ver también Web Service HTTP Error code) 9913: Error red 9914: Error interno.
Web Service fault actor	Entero largo	3	Causa del error (devuelto por el protocolo SOAP, a utilizar en caso de error principal 9910). - Version Mismatch - Must Understand (un parámetro definido como obligatorio no puede ser interpretado por el servidor) - Sender Fault - Receiver Fault - Encoding Unknown
Web Service HTTP compression	Entero largo	6	valor = Web Service Compression Esta opción permite activar un mecanismo interno de compresión de las peticiones SOAP con el fin de acelerar los intercambios entre aplicaciones 4D. Cuando ejecuta la instrucción **WEB SERVICE SET OPTION**(Web Service HTTP Compression; Web Service Compression) en el cliente 4D del servicio web, los datos de la próxima petición SOAP enviados por el cliente serán comprimidos utilizando un mecanismo estándar HTTP ("gzip" o "deflate" en función del contenido de la petición) antes de su envío al servidor SOAP 4D. El servidor descomprimirá y analizará la petición, luego responderá automáticamente utilizando el mismo mecanismo. Sólo se afecta la petición que sigue la llamada al comando **WEB SERVICE SET OPTION**. Por lo tanto debe llamar este comando cada vez que quiera utilizar la compresión. Por defecto, 4D no comprime las peticiones HTTP de los servicios web. **Nota:** este mecanismo no puede utilizarse para las peticiones enviadas a un servidor SOAP 4D de una versión anterior a la 11.3. Para que pueda optimizar más este funcionamiento, las opciones adicionales configuran el límite y la tasa de compresión de las peticiones. Estas opciones son accesibles vía el comando **SET DATABASE PARAMETER**.
Web Service HTTP error code	Entero largo	2	Código del error HTTP (a utilizar en caso de error principal 9912).
Web Service HTTP timeout	Entero largo	1	valor = "timeout" de la parte cliente expresado en segundos. El timeout de la parte clientes es el periodo de espera del cliente servicio web en caso de que no haya respuesta del servidor. Después de este período, el cliente cierra la sesión y se pierde la petición. Por defecto, este timeout es de 180 segundos. Puede modificarse por razones específicas (estado de la red, especificaciones del servicio web, etc.).
Web Service manual	Entero largo	3	
Web Service manual in	Entero largo	1	
Web Service manual out	Entero largo	2	
Web Service reset auth settings	Entero largo	5	valor = 0 (no borrar la información) ó 1 (borra la información) Esta opción le permite indicar a 4D memorizar la información de autenticación del usuario (nombre de usuario, contraseña, método, etc.), para reutilizarla posteriormente. Por defecto, esta información se borra después de cada ejecución del comando **CALL WEB SERVICE**. Pase 0 en valor para guardar la información y 1 para borrarla. Note que cuando pasa 0, la información se conserva durante la sesión pero no se almacena.

Constante	Tipo	Valor	Comentario
Web Service SOAP header	*Entero largo*	*2*	*valor = referencia del elemento XML raíz a insertar como encabezado de la petición SOAP. Esta opción permite insertar un encabezado en la petición SOAP generada utilizando el comando **CALL WEB SERVICE**. Por defecto, las peticiones SOAP no contienen un encabezado específico. Sin embargo, algunos servicios web requieren un encabezado, por ejemplo para la gestión de los parámetros de identificación.*
Web Service SOAP version	*Entero largo*	*3*	*valor = Web Service SOAP_1_1 o Web Service SOAP_1_2 Esta opción permite precisar la versión del protocolo SOAP utilizado en la petición. Pase en valor la constante Web Service SOAP_1_1 para indicar la versión 1.1 y la constante Web Service SOAP_1_2 para indicar la versión 1.2.*
Web Service SOAP_1_1	*Entero largo*	*0*	
Web Service SOAP_1_2	*Entero largo*	*1*	

Servicios Web (Servidor)

Constante	*Tipo*	*Valor*	*Comentario*
Is DOM reference	*Entero largo*	*37*	
Is XML	*Entero largo*	*36*	
SOAP client fault	*Entero largo*	*1*	
SOAP input	*Entero largo*	*1*	
SOAP method name	*Entero largo*	*1*	*Nombre del método ofrecido como servicio web a punto de ejecutarse*
SOAP output	*Entero largo*	*2*	
SOAP server fault	*Entero largo*	*2*	
SOAP service name	*Entero largo*	*2*	*Nombre del servicio web al que pertenece el método*

Servidor web

Constante	Tipo	Valor	Comentario
wdl disable	Entero largo	0	El archivo de historial de peticiones HTTP Web está desactivado
wdl enable with all body parts	Entero largo	7	El archivo de historial de peticiones HTTP Web está activado con el cuerpo de la respuesta y la respuesta
wdl enable with request body	Entero largo	5	El archivo de historial de peticiones HTTP Web está activado con el cuerpo de la respuesta únicamente
wdl enable with response body	Entero largo	3	El archivo de historial de peticiones HTTP Web está activado con el cuerpo de la respuesta únicamente
wdl enable without body	Entero largo	1	El archivo de historial de peticiones HTTP Web está desactivado sin el cuerpo (el tamaño del cuerpo se entrega en este caso)
Web character set	Entero largo	17	**Alcance**: 4D local, 4D Server **Descripción**: conjunto de caracteres que el servidor Web 4D (con 4D en modo local y 4D Server) utiliza para comunicarse con los navegadores web que se conectan a base. El valor por defecto depende del lenguaje del sistema operativo. Este parámetro se define en las Propiedades de la base. **Valores**: los valores posibles dependen del modo de ejecución de la base relativos al conjunto de caracteres. • Modo Unicode: cuando la aplicación se ejecuta en modo Unicode, los valores a pasar para este parámetro son los identificadores de conjunto de caracteres (MIBEnum longint o Name string, identificadores definidos por IANA, consulte: http://www.iana.org/assignments/character-sets). Está es la lista de los identificadores correspondientes a los conjuntos de caracteres que admite el servidor Web de 4D: 4=ISO-8859-1 12=ISO-8859-9 13=ISO-8859-10 17=Shift-JIS 2024=Windows-31J 2026=Big5 38=euc-kr 106=UTF-8 2250=Windows-1250 2251=Windows-1251 2253=Windows-1253 2255=Windows-1255 2256=Windows-1256 • Modo compatibilidad ASCII: 0: Occidental 1: Japonés 2: Chino 3: Coreano 4: Definido por el usuario 5: Reservado 6: Europa central 7: Cirílico 8: Árabe 9: Griego 10: Hebreo 11: Turco 12: Nórdico
Web Client IP address to listen	Entero largo	23	**Alcance**: todas las máquinas remotas 4D **Valores posibles**: ver Web IP address to listen **Descripción**: se utiliza para especificar este parámetro para todas las máquinas 4D remotas utilizadas como servidores web (aplicado al servidor web remoto únicamente).
Web debug log	Entero largo	84	**Alcance:** servidor web local **Nota:** si se reinicia el servidor HTTP, se utiliza un nuevo archivo de historial **Descripción:** le permite obtener o definir el estado del archivo de historial de peticiones HTTP del servidor Web 4D. Cuando se activa, este archivo, llamado "**HTTPDebugLog_nn.txt**", se guarda en la carpeta "Logs" de la aplicación (nn es el número de archivo). Es útil para problemas de depuración relacionados con el servidor web. Registra cada petición y cada respuesta en modo raw. La totalidad de las peticiones, encabezados, se registran; opcionalmente, también se pueden registrar partes del cuerpo. Para mayor información sobre archivos HTTPDebugLog, consulte la sección **Anexo E: Descripción de archivos de historial**. **Valores**: una de las constantes con el prefijo "wdl" (consulte las descripciones de estas constantes en este tema). **Valor por defecto**: 0 (no activado)

Constante	Tipo	Valor	Comentario
Web HSTS enabled	Entero largo	86	**Alcance**: 4D local, 4D Server. **Descripción**: estado HTTP Strict Transport Security (HSTS). HSTS permite que el servidor Web 4D declare que los navegadores solo deberían interactuar con él a través de conexiones HTTPS seguras. Una vez activado, el servidor web 4D agregará automáticamente información relacionada con HSTS a todos los encabezados de respuesta.Los navegadores registrarán la información HSTS la primera vez que reciban una respuesta del servidor web 4D, luego toda solicitud HTTP futura se transformará automáticamente en solicitudes HTTPS. El tiempo que el navegador almacena esta información se especifica con el selector Web HSTS max age. HSTS requiere que HTTPS esté habilitado en el servidor. HTTP también debe estar habilitado para permitir las conexiones iniciales del cliente. **Valores posibles**: 0 (desactivado, predeterminado) o 1 (activado) **Nota:** el servidor web 4D debe reiniciarse para que se aplique esta configuración.
Web HSTS max age	Entero largo	87	**Alcance**: 4D local, 4D Server **Descripción**: especifica el tiempo máximo (en segundos) que HSTS está activo para cada conexión de cliente nuevo. Esta información se almacena en el lado del cliente durante la duración especificada. **Valores posibles**: Entero largo (segundos) **Valor por defecto**: 63072000 (2 años) **Atención:** una vez que HSTS esté habilitado, las conexiones cliente continuarán usando este mecanismo por la duración especificada. Cuando esté probando sus aplicaciones, se recomienda establecer una duración corta para poder cambiar entre los modos de conexión segura y no segura si es necesario.
Web HTTP compression level	Entero largo	50	**Alcance**: Servidor web local **Descripción**: nivel de compresión para todos los intercambios HTTP comprimidos efectuados para el servidor HTTP de 4D (peticiones cliente o respuestas servidor, Web y servicio web). Este selector permite optimizar los intercambios con un enfoque en la velocidad de ejecución (menor compresión) o la cantidad de compresión (menor velocidad). La elección de un valor depende del tamaño y la naturaleza de los datos intercambiados. Pase de 1 a 9 en el parámetro valor, 1 es la compresión más rápida y 9 la más alta. También puede pasar -1 para obtener un compromiso entre velocidad y tasa de compresión. El nivel de compresión por defecto es 1 (compresión rápida). **Valores posibles**: 1 a 9 (1 = más rápido, más comprimido = 9) o -1 = mejor compromiso.
Web HTTP compression threshold	Entero largo	51	**Alcance**: Servidor HTTP local **Descripción**: en intercambios HTTP optimizados, límite de tamaño de petición por debajo del cual los intercambios no deben comprimirse. Esta opción es útil para evitar pérdidas de tiempo de máquina para comprimir intercambios muy pequeños. Pase en valor un tamaño en bytes. Por defecto, el límite de compresión se establece en 1024 bytes. **Valores posibles**: todo valor de tipo entero largo. El parámetro valor contiene una tamaño expresado en bytes. Por defecto, el umbral de compresión está definido en 1024 bytes.
Web HTTP enabled	Entero largo	88	**Alcance**: 4D local, 4D Server **Descripción**: estados para comunicación sobre HTTP. **Valores posibles**: 0 (desactivado) o 1 (activado)
Web HTTP TRACE	Entero largo	85	**Alcance**: servidor web local **Descripción**: le permite activar o desactivar el método HTTP TRACE en el servidor web de 4D. Por razones de seguridad, a partir de 4D v15 R2, por defecto, el servidor web 4D rechaza peticiones TRACE HTTP con un error 405 (ver desactivación de HTTP TRACE). Si es necesario, puede activar el método HTTP TRACE para la sesión pasando esta constante con el valor 1. Cuando se activa esta opción, el servidor web 4D responde a las solicitudes HTTP TRACE con la línea de petición, el encabezado y el cuerpo. **Valores posibles**: 0 (desactivado) o 1 (activado) **Valor por defecto**: 0 (desactivado)
Web HTTPS enabled	Entero largo	89	**Alcance**: 4D local, 4D Server **Descripción**: estado para comunicación sobre HTTPS. **Valores posibles**: 0 (desactivado) o 1 (activado)
Web HTTPS port ID	Entero largo	39	**Alcance**: 4D local, 4D Server **Valores posibles**: 0 a 65535 **Descripción**: número del puerto TCP utilizado por el servidor web de 4D en modo local y de 4D Server para conexiones seguras vía TLS (protocolo HTTPS). El número de puerto HTTPS se define en la página "Web/Configuración" de la caja de diálogo Propiedades de la base. Por defecto, el valor es 443 (valor estándar). Puede utilizar las constantes del tema **Números de puerto TCP** para el parámetro valor.
Web inactive process timeout	Entero largo	78	**Alcance**: servidor web local **Descripción**: permite modificar el timeout del proceso utilizado para la sesión (opción relativa al proceso). Después del timeout, el proceso se elimina en el servidor, se llama al **Método base On Web Close Process** y luego el contexto de la sesión se destruye. **Valores**: Entero largo (minutos) **Valores por defecto**: 480 minutos (pase 0 para restablecer el valor por defecto)
Web inactive session timeout	Entero largo	72	**Alcance**: servidor web local **Descripción**: permite modificar la duración de vida de las sesiones inactivas (duración definida en cookie). Al final de este periodo, la cookie de sesión expira y no se envía más al cliente HTTP. **Valores**: Entero largo (minutos) **Valores por defecto**: 480 minutos (pase 0 para restablecer el valor por defecto)

Constante	Tipo	Valor	Comentario
Web IP address to listen	*Entero largo*	*16*	***Alcance**: 4D local, 4D Server* ***Descripción**: dirección IP en la que el servidor web debe recibir las peticiones HTTP con 4D en modo local y 4D Server. Por defecto, ninguna dirección se especifica. Este parámetro se define en las Propiedades de la base. Este selector es útil en el caso de los servidores web 4D compilados y fusionados con 4D Desktop (no hay acceso al modo Diseño).* ***Valores posibles**: dirección IP en forma de cadena. Ambos formatos cadena IPv6 (por ejemplo, "2001:0db8:0000:0000:0000:ff00:0042:8329") y los formatos cadena IPv4 (por ejemplo, "123.45.67.89") son compatibles.* ***Nota:** por compatibilidad, las direcciones IPv4 expresadas en como longitudes hexadecimales (obsoletas) aún son compatibles.*
Web keep session	*Entero largo*	*70*	***Alcance**: servidor web local* ***Descripción**: permite activar o desactivar el modo de gestión de las sesiones (descrito en la sección **Gestión de las sesiones web**).* ***Valores**: 1 (activar modo) ó 0 (desactivar modo)* ***Valor por defecto**: 1 para bases creadas en la versión 13, 0 para bases convertidas. Note que este modo activa igualmente el mecanismo de reutilización de los contextos temporales en modo remoto. Para mayor información sobre este mecanismo, consulte la descripción de esta opción en la sección **Parámetros del servidor web**.*
Web log recording	*Entero largo*	*29*	***Alcance**: 4D local 4D Server* ***Descripción**: inicia o detiene el registro de peticiones solicitudes Web recibida por el servidor web de 4D en modo local o 4D Server. Por defecto, el valor es 0 (no hay registro de peticiones).* *El historial de las peticiones web se guarda en un archivo texto llamado "logweb.txt" que se ubica automáticamente en la carpeta Logs de la base, junto al archivo de estructura. El formato de este fichero es determinado por el valor que se pase. Para más información sobre los diferentes formatos de historial de las peticiones, consulte la sección [#title id= "2833"/]. La activación de este archivo también se puede definir en la página "Web/Avanzado" de las Preferencias de 4D.* ***Valores posibles**: 0 = No guardar (por defecto), 1 = Registrar en formato CLF, 2 = Registrar en formato DLF, 3 = Registrar en formato DLF, 4 = Guardar en formato WLF.* ***Atención**: los formatos 3 y 4 formatos son formatos personalizados, los contenidos deben ser definidos de antemano en las Preferencias de la aplicación, página "Web/Formato del historial". Si usted utiliza uno de estos formatos sin que sus campos hayan sido seleccionados, el archivo de las peticiones no se generará.*
Web max concurrent processes	*Entero largo*	*18*	***Alcance**: 4D local, 4D Server* ***Descripción**: límite estrictamente superior del número de procesos web de todo tipo aceptados por el servidor web con 4D en modo local y 4D Server. Cuando se alcanza el número límite (menos uno), 4D no crea un nuevo proceso y devuelve el mensaje "Servidor no disponible" (estado HTTP 503 - Servicio no disponible) a toda nueva petición.* *Este parámetro previene la saturación del servidor web de 4D que puede ocurrir durante un envío masivo de peticiones o de una demanda excesiva de creación de contextos. También puede definirse en la caja de diálogo de la Propiedades de la base.* *En teoría, el número máximo de procesos web es el resultado de dividir la memoria disponible / tamaño de la pila de un proceso web. Otra solución es ver la información sobre los procesos web que se muestra en el Explorador de ejecución: se indican el número actual de procesos web y el número máximo alcanzado desde el inicio del servidor web.* ***Valores**: todo valor entre 10 y 32 000. El valor por defecto es 100.*
Web max sessions	*Entero largo*	*71*	***Alcance**: servidor web local* ***Descripción**: permite limitar el número de sesiones simultáneas. Cuando se alcanza el número definido, la sesión más antigua se cierra (y se llama al **Método base On Web Close Process**) si el servidor web necesita crear una nueva.* ***Valores posibles:** Entero largo. El número de sesiones simultáneas no puede superar el número total de procesos web (opción Web Max Concurrent Processes, 100 por defecto)* ***Valores por defecto**: 100 (pase 0 para restablecer el valor por defecto)*
Web maximum requests size	*Entero largo*	*27*	***Alcance**: 4D local, 4D Server* ***Descripción**: tamaño máximo (en bytes) de las peticiones HTTP entrantes (POST) que el servidor web está autorizado a tratar. Por defecto, el valor predeterminado es 2 000 000, es decir, un poco menos de 2 MB. El valor máximo (2 147 483 648) significa en la práctica que ningún límite se establece.* *Esta configuración evita la saturación del servidor web, causadas por peticiones entrantes muy grandes. Cuando una petición llega al límite, el servidor web de 4D la rechaza.* ***Valores posibles**: 500 000 a 2 147 483 648.*
Web port ID	*Entero largo*	*15*	***Alcance**: 4D en modo local y 4D Server.* ***Descripción**: establece u obtiene el número del puerto TCP utilizado por el servidor web 4D con 4D en modo local y 4D Server. Por defecto, el valor es 80. El número de puerto TCP se define en la página "Web/Configuración" de la caja de diálogo Propiedades de la base. Puede utilizar una de las constantes del tema **Números de puerto TCP** para el parámetro valor. Este selector es útil en el marco de servidores web 4D que se compilan y fusionan utilizando 4D de escritorio (sin acceso al entorno Diseño).* ***Valores posibles**: para obtener más información sobre el número de puerto TCP, consulte la sección **Parámetros del servidor web**.* ***Valor por defecto**: 80*
Web session cookie domain	*Entero largo*	*81*	***Alcance**: Servidor web local* ***Descripción**: define u obtiene el valor del campo "dominio" de la cookie de sesión. Este selector (así como el selector 82) es útil para controlar el alcance de las cookies de sesión: si configura, por ejemplo, el valor "/*.4d.fr" para este selector, el cliente sólo enviará una cookie cuando la petición se dirige al dominio ".4d.fr", que excluye los servidores que alojan los datos estáticos externos.* ***Valores posibles**: Texto*

Constante	Tipo	Valor	Comentario
Web session cookie name	*Entero largo*	*73*	***Alcance**: servidor web local* ***Descripción**: permite definir el nombre de la cookie utilizada para almacenar el ID de la sesión.* ***Valores**: Texto* ***Valores por defecto**: "4DSID" (pase una cadena vacía para restablecer el valor por defecto)*
Web session cookie path	*Entero largo*	*82*	***Alcance**: servidor web local* ***Descripción**: define u obtiene el valor del campo "path" de la cookie de sesión. Este selector (así como elselector 81) es útil para controlar el alcance de las cookies de sesión: si configura, por ejemplo, el valor "/4DACTION" para este selector, el cliente deberá enviar sólo una cookie para peticiones dinámicas que comiencen con 4DACTION , y no para las imágenes, páginas estáticas, etc.* ***Valores posibles**: Texto*
Web session enable IP address validation	*Entero largo*	*83*	***Alcance**: servidorWeb Local* ***Descripción**: Activa o desactiva la validación de las direcciones IP para las cookies de sesión. Por razones de seguridad, por defecto, el servidor web de 4D verifica la dirección IP de cada solicitud que contiene una cookie de sesión y la rechaza si esta dirección no coincide con la dirección IP utilizada para crear la cookie. En algunas aplicaciones específicas, es posible que desee desactivar esta validación y aceptar las cookies de sesión, incluso cuando sus direcciones IP no coincidan. Por ejemplo, cuando los dispositivos móviles cambian entre redes WiFi y 3G/4G, su dirección IP cambia. En este caso, debe pasar 0 en esta opción para permitir que los clientes puedan seguir utilizando sus sesiones web incluso cuando las direcciones IP cambien. Tenga en cuenta que esta configuración reduce el nivel de seguridad de la aplicación.* *Cuando se modifica, esta configuración es efectiva inmediatamente (no es necesario reiniciar el servidor HTTP).* ***Valores posibles**: 0 (desactivado) o 1 (activado)* ***Valor por defecto**: 1 (las direcciones IP son verificadas)*

SQL

Constante	Tipo	Valor	Comentario
SQL all records	Entero largo	-1	
SQL asynchronous	Entero largo	1	0 = Conexión sincrónica (valor por defecto), 1 (o valor diferente de 0) = Conexión asincrónica
SQL charset	Entero largo	100	Codificación del texto utilizada por las peticiones enviadas a fuentes externas (vía SQL pass-through). La modificación se lleva a cabo para el proceso actual y la conexión actual. Valores posibles: identificador MIBEnum (ver nota 2), cadena "WCHAR" (ver nota 3) o valor -2 (ver nota 4). Por defecto: 106 (UTF-8)
SQL connection timeout	Entero largo	5	Tiempo máximo de espera durante la ejecución del comando **SQL LOGIN**. Este valor debe definirse antes de abrir la conexión para que sea tenido en cuenta. Valores posibles: tiempo en segundos Por defecto: no hay timeout
SQL max data length	Entero largo	3	Longitud máxima de los datos devueltos
SQL max rows	Entero largo	2	Número máximo de líneas en el conjunto resultante (utilizado para previsualizaciones)
SQL On error abort	Entero largo	1	En caso de error, 4D detiene de inmediato la ejecución del script.
SQL On error confirm	Entero largo	2	En caso de error, 4D muestra una caja de diálogo que describe el error y permite al usuario interrumpir o continuar la ejecución del script.
SQL On error continue	Entero largo	3	En caso de error, 4D lo ignoras y continua la ejecución del script.
SQL param in	Entero largo	1	
SQL param in out	Entero largo	2	Utilizable únicamente en el contexto de un procedimiento almacenado SQL (entrada-salida parámetro definido en el procedimiento almacenado)
SQL param out	Entero largo	4	Utilizable únicamente en el contexto de un procedimiento almacenado SQL (parámeto salida definido en el procedimiento almacenado)
SQL query timeout	Entero largo	4	Tiempo máximo de espera al ejecutar el comando **SQL EXECUTE**. Valores: tiempo en segundos Por defecto: no hay timeout
SQL use access rights	Cadena	SQL_Use_Access_Rights	Permite restringir los derechos de acceso a aplicar durante la ejecución de los comandos SQL del script. Cuando utilice este atributo, debe pasar 0 ó 1 en attribValue: • attribValue = 1: 4D utiliza los derechos de acceso del usuario 4D actual. • attribValue = 0 (o atributo no definido): 4D no restringe el acceso, se utilizan los derechos del Diseñador.
SQL_INTERNAL	Cadena	;DB4D_SQL_LOCAL;	
System data source	Entero largo	2	
User data source	Entero largo	1	

Teclas de función

Constante	Tipo	Valor	Comentario
Backspace key	Entero largo	8	
Down arrow key	Entero largo	31	
End key	Entero largo	4	
Enter key	Entero largo	3	
Escape key	Entero largo	27	
F1 key	Entero largo	-122	
F10 key	Entero largo	-109	
F11 key	Entero largo	-103	
F12 key	Entero largo	-111	
F13 key	Entero largo	-105	
F14 key	Entero largo	-107	
F15 key	Entero largo	-113	
F2 key	Entero largo	-120	
F3 key	Entero largo	-99	
F4 key	Entero largo	-118	
F5 key	Entero largo	-96	
F6 key	Entero largo	-97	
F7 key	Entero largo	-98	
F8 key	Entero largo	-100	
F9 key	Entero largo	-101	
Help key	Entero largo	5	
Home key	Entero largo	1	
Left arrow key	Entero largo	28	
Page down key	Entero largo	12	
Page up key	Entero largo	11	
Return key	Entero largo	13	
Right arrow key	Entero largo	29	
Tab key	Entero largo	9	
Up arrow key	Entero largo	30	

Texto multiestilo

Constante	Tipo	Valor	Comentario
ST 4D Expressions as sources	Entero largo	2	Se devuelve la cadena original de las referencias de expresiones 4D
ST 4D Expressions as values	Entero largo	1	Las referencias de expresiones 4D se devuelven en su forma evaluada (funcionamiento por defecto en los formularios)
ST End highlight	Entero largo	-1001	Designa el último carácter de la selección actual de texto en el objeto (*)
ST End text	Entero largo	0	Designa el último carácter del texto contenido en el objeto
ST Expression type	Entero largo	2	La selección contiene sólo una referencia de expresión
ST Expressions display mode	Entero largo	1	El parámetro valor puede contener ST Values o ST References
ST Mixed type	Entero largo	3	La selección contiene al menos dos tipos de contenidos diferentes
ST Picture type	Entero largo	6	La selección contiene sólo una imagen (áreas 4D Write Pro únicamente)
ST Plain type	Entero largo	0	La selección contiene texto y ninguna referencia
ST References	Entero largo	1	Muestra las cadenas fuente de las expresiones
ST References as spaces	Entero largo	0	Cada referencia se devuelve como un carácter espacio sin separación (funcionamiento por defecto, utilizado por los otros comandos)
ST Start highlight	Entero largo	-1000	Designa el primer carácter de la selección actual de texto en el objeto (*)
ST Start text	Entero largo	1	Designa el primer carácter del texto contenido en el objeto
ST Tags as plain text	Entero largo	64	El rótulo de la etiqueta se devuelve en texto plano. Por ejemplo para el tag '<img src="test.jpg" alt="picture">my picture</img>', el texto plano es "my picture" (funcionamiento por defecto en los formularios)
ST Tags as XML code	Entero largo	128	El código XML de la etiqueta se devuelve en texto plano. Por ejemplo para el tag '<img src="test.jpg" alt="picture">my picture</img>', el texto plano es '<img src="test.jpg" alt="picture">my picture</img>'
ST Text displayed with 4D Expression sources	Entero largo	86	Devuelve el texto tal y como se muestra en los formularios con la cadena de origen de las expresiones 4D. Corresponde a la combinación predefinida de las constantes 2+4+16+64.
ST Text displayed with 4D Expression values	Entero largo	85	Devuelve el texto tal y como se muestra en los formularios con las expresiones 4D en su forma evaluada. Corresponde a la combinación predefinida de las constantes 1+4+16+64.
ST Unknown tag type	Entero largo	4	La selección contiene sólo una etiqueta de tipo desconocido
ST URL as labels	Entero largo	4	La etiqueta visible de los URLs se devuelve, por ejemplo "Visite nuestro sitio web" (funcionamiento por defecto en los formularios)
ST URL as links	Entero largo	8	Se devuelve el enlace, por ejemplo "http://www.4d.com"
ST URL type	Entero largo	1	La selección contiene sólo una referencia de URL
ST User links as labels	Entero largo	16	Se devuelve la etiqueta visible del enlace usuario (funcionamiento por defecto en los formularios)
ST User links as links	Entero largo	32	Se devuelve el contenido del enlace usuario
ST User type	Entero largo	5	La selección contiene sólo una referencia personalizada
ST Values	Entero largo	0	Muestra los valores calculados de las expresiones

Tipo de índice

Constante	*Tipo*	*Valor*	*Comentario*
Cluster BTree index	*Entero largo*	*3*	*Índice de tipo B-Tree utilizando clusters. Este tipo de índice se optimiza cuando el índice contiene pocas palabras claves, es decir cuando los mismos valores se presentan con frecuencia en los datos.*
Default index type	*Entero largo*	*0*	*4D define el tipo de índice (excepto los índices de palabras claves) que es el más optimizado en función del contenido del campo.*
Keywords index	*Entero largo*	*-1*	*Permite la indexación palabra por palabra del contenido del campo. Este tipo de índice sólo puede utilizarse con campos de tipo Texto, Alfa e Imagen. Atención los índices de palabras claves no pueden ser compuestos*
Standard BTree index	*Entero largo*	*1*	*Índice de tipo B-Tree clásico. Este tipo de índice multi propósito se utiliza en las versiones anteriores de 4D*

Tipo de lista de las fuentes

Constante	*Tipo*	*Valor*	*Comentario*
Favorite fonts	*Entero largo*	*1*	*fuentes contiene la lista de fuentes favoritas.* *- Bajo de Windows: lista de nombres de familias de fuentes activas en el panel de control de Windows.* *- Bajo OS X: lista de nombres de familias de fuentes de la colección "com.apple.Favorites" que se encuentra en el panel de control, llamada "Favorites" en Inglés, "Favoris" en francés, "Favoriten" en alemán, etc. Esta colección puede estar en blanco si el usuario no ha añadido fuentes favoritas.*
Recent fonts	*Entero largo*	*2*	*fuentes contiene la lista de fuentes recientes (lista de fuentes utilizadas durante la sesión 4D). Esta lista es utilizada particularmente por las áreas de texto multiestilo.*
System fonts	*Entero largo*	*0*	*fuentes contiene la lista de todas las fuentes del sistema. Opción por defecto si se omite tipoLista.*

Tipo de proceso

Constante	Tipo	Valor	Comentario
_o_Web process with context	Entero largo	-11	
Apple event manager	Entero largo	-7	
Backup process	Entero largo	-19	
Cache manager	Entero largo	-4	
Client manager process	Entero largo	-31	
Compiler process	Entero largo	-29	
Created from execution dialog	Entero largo	3	
Created from menu command	Entero largo	2	
DB4D Cron	Entero largo	-49	
DB4D Flush cache	Entero largo	-46	
DB4D Garbage collector	Entero largo	-47	
DB4D Index builder	Entero largo	-45	
DB4D Listener	Entero largo	-51	
DB4D Mirror	Entero largo	-50	
DB4D Worker pool user	Entero largo	-48	
Design process	Entero largo	-2	
Event manager	Entero largo	-8	
Execute on client process	Entero largo	-14	
Execute on server process	Entero largo	1	
External task	Entero largo	-9	
HTTP Listener	Entero largo	-56	
HTTP Log flusher	Entero largo	-58	
HTTP Worker pool server	Entero largo	-55	
Indexing process	Entero largo	-5	
Internal 4D server process	Entero largo	-18	
Internal timer process	Entero largo	-25	
Log file process	Entero largo	-20	
Logger process	Entero largo	-57	
Main 4D process	Entero largo	-39	
Main process	Entero largo	-1	
Method editor macro process	Entero largo	-17	
Monitor process	Entero largo	-26	
MSC process	Entero largo	-22	
None	Entero largo	0	
On exit process	Entero largo	-16	
Other 4D process	Entero largo	-10	
Other internal process	Entero largo	-40	
Other user process	Entero largo	4	
Restore Process	Entero largo	-21	
Serial Port Manager	Entero largo	-6	
Server interface process	Entero largo	-15	
ServerNet Listener	Entero largo	-43	
ServerNet Session manager	Entero largo	-44	
SQL Listener	Entero largo	-54	
SQL Method execution process	Entero largo	-24	
SQL Net Session manager	Entero largo	-53	
SQL Worker pool server	Entero largo	-52	
Web process on 4D remote	Entero largo	-12	
Web process with no context	Entero largo	-3	
Web server process	Entero largo	-13	
Worker pool in use	Entero largo	-41	
Worker pool spare	Entero largo	-42	
Worker process	Entero largo	5	Procesos Worker lanzado por el usuario

Tipo Digest

Constante	*Tipo*	*Valor*	*Comentario*
4D digest	*Entero largo*	*2*	*Utilizar el algoritmo interno de 4D*
MD5 digest	*Entero largo*	*0*	*Utilizar el algoritmo MD5*
SHA1 digest	*Entero largo*	*1*	*Utilizar el algoritmo SHA-1*
SHA256 digest	*Entero largo*	*3*	*(Familia SHA-2) SHA-256 es una serie de 256 bits devueltos como una cadena de 64 caracteres hexadecimales.*
SHA512 digest	*Entero largo*	*4*	*(Familia SHA-2) SHA-512 es una serie de 512 bits devueltos como una cadena de 128 caracteres hexadecimales.*

Tipos de campos y variables

Constante	Tipo	Valor	Comentario
Array 2D	Entero largo	13	
Blob array	Entero largo	31	
Boolean array	Entero largo	22	
Date array	Entero largo	17	
Integer array	Entero largo	15	
Is alpha field	Entero largo	0	
Is BLOB	Entero largo	30	
Is Boolean	Entero largo	6	
Is collection	Entero largo	42	
Is date	Entero largo	4	
Is float	Entero largo	35	
Is integer	Entero largo	8	
Is integer 64 bits	Entero largo	25	
Is longint	Entero largo	9	
Is null	Entero largo	255	
Is object	Entero largo	38	
Is picture	Entero largo	3	
Is pointer	Entero largo	23	
Is real	Entero largo	1	
Is string var	Entero largo	24	
Is subtable	Entero largo	7	
Is text	Entero largo	2	
Is time	Entero largo	11	
Is undefined	Entero largo	5	
LongInt array	Entero largo	16	
Object array	Entero largo	39	
Picture array	Entero largo	19	
Pointer array	Entero largo	20	
Real array	Entero largo	14	
String array	Entero largo	21	
Text array	Entero largo	18	
Time array	Entero largo	32	

Tipos objetos formulario

Constante	*Tipo*	*Valor*	*Comentario*
Object type 3D button	*Entero largo*	*16*	
Object type 3D checkbox	*Entero largo*	*26*	
Object type 3D radio button	*Entero largo*	*23*	
Object type button grid	*Entero largo*	*20*	
Object type checkbox	*Entero largo*	*25*	
Object type combobox	*Entero largo*	*11*	
Object type dial	*Entero largo*	*28*	
Object type group	*Entero largo*	*21*	
Object type groupbox	*Entero largo*	*30*	
Object type hierarchical list	*Entero largo*	*6*	
Object type hierarchical popup menu	*Entero largo*	*13*	
Object type highlight button	*Entero largo*	*17*	
Object type invisible button	*Entero largo*	*18*	
Object type line	*Entero largo*	*32*	
Object type listbox	*Entero largo*	*7*	
Object type listbox column	*Entero largo*	*9*	
Object type listbox footer	*Entero largo*	*10*	
Object type listbox header	*Entero largo*	*8*	
Object type matrix	*Entero largo*	*35*	
Object type oval	*Entero largo*	*34*	
Object type picture button	*Entero largo*	*19*	
Object type picture input	*Entero largo*	*4*	
Object type picture popup menu	*Entero largo*	*14*	
Object type picture radio button	*Entero largo*	*24*	
Object type plugin area	*Entero largo*	*38*	
Object type popup dropdown list	*Entero largo*	*12*	
Object type progress indicator	*Entero largo*	*27*	
Object type push button	*Entero largo*	*15*	
Object type radio button	*Entero largo*	*22*	
Object type radio button field	*Entero largo*	*5*	
Object type rectangle	*Entero largo*	*31*	
Object type rounded rectangle	*Entero largo*	*33*	
Object type ruler	*Entero largo*	*29*	
Object type splitter	*Entero largo*	*36*	
Object type static picture	*Entero largo*	*2*	
Object type static text	*Entero largo*	*1*	
Object type subform	*Entero largo*	*39*	
Object type tab control	*Entero largo*	*37*	
Object type text input	*Entero largo*	*3*	
Object type unknown	*Entero largo*	*0*	
Object type view pro area	*Entero largo*	*42*	
Object type web area	*Entero largo*	*40*	
Object type write pro area	*Entero largo*	*41*	

Transformación de imágenes

Constante	*Tipo*	*Valor*	*Comentario*
Crop	*Entero largo*	*100*	
Fade to grey scale	*Entero largo*	*101*	
Flip horizontally	*Entero largo*	*3*	
Flip vertically	*Entero largo*	*4*	
Horizontal concatenation	*Entero largo*	*1*	
Reset	*Entero largo*	*0*	
Scale	*Entero largo*	*1*	
Superimposition	*Entero largo*	*3*	
Translate	*Entero largo*	*2*	
Transparency	*Entero largo*	*102*	
Vertical concatenation	*Entero largo*	*2*	

Valores para acción estándar asociada

Todas las constantes de este tema están obsoletas a partir de 4D v16 R3. Debe utilizar las constantes de texto del tema ***Acción estándar****.*

Constante	Tipo	Valor	Comentario
_o_Accept action	Entero largo	2	
_o_Add subrecord action	Entero largo	14	
_o_Cancel action	Entero largo	1	
_o_Clear action	Entero largo	21	
_o_Copy action	Entero largo	19	
_o_Cut action	Entero largo	18	
_o_Database settings action	Entero largo	32	
_o_Delete record action	Entero largo	7	
_o_Delete subrecord action	Entero largo	13	
_o_Edit subrecord action	Entero largo	12	
_o_First page action	Entero largo	10	
_o_First record action	Entero largo	5	
_o_Last page action	Entero largo	11	
_o_Last record action	Entero largo	6	
_o_MSC action	Entero largo	36	
_o_Next page action	Entero largo	8	
_o_Next record action	Entero largo	3	
_o_No action	Entero largo	0	
_o_Paste action	Entero largo	20	
_o_Previous page action	Entero largo	9	
_o_Previous record action	Entero largo	4	
_o_Quit action	Entero largo	27	
_o_Redo action	Entero largo	31	
_o_Return to Design mode	Entero largo	35	
_o_Select all action	Entero largo	22	
_o_Show clipboard action	Entero largo	23	
_o_Test application action	Entero largo	26	
_o_Undo action	Entero largo	17	

Ventana

Constante	*Tipo*	*Valor*	*Comentario*
External window	*Entero largo*	*5*	
Floating window	*Entero largo*	*14*	
Modal dialog	*Entero largo*	*9*	
Regular window	*Entero largo*	*8*	
XY Current form	*Entero largo*	*1*	*El origen es la esquina superior izquierda del formulario actual*
XY Current window	*Entero largo*	*2*	*El origen es la esquina superior izquierda de la ventana actual*
XY Main window	*Entero largo*	*4*	*En Windows: origen es la esquina superior izquierda de la ventana principal; en OS X: igual que XY Screen*
XY Screen	*Entero largo*	*3*	*El origen es la esquina superior izquierda de la pantalla principal (igual que para el comando **SCREEN COORDINATES**)*

Web Area

Constante	*Tipo*	*Valor*	*Comentario*
WA enable contextual menu	*Entero largo*	*4*	*Autoriza la visualización del menú contextual estándar en el área web*
WA enable Java applets	*Entero largo*	*1*	*Autoriza la ejecución de applets Java en el área web.*
WA enable JavaScript	*Entero largo*	*2*	*Autoriza la ejecución de JavaScript en el área web*
WA enable plugins	*Entero largo*	*3*	*Autoriza la instalación de plug-ins en el área web*
WA enable URL drop	*Entero largo*	*101*	*Permite soltar URLs o archivos en el área web (por defecto = false)*
WA enable Web inspector	*Entero largo*	*100*	*Permite la visualización del inspector web en el área*
WA next URLs	*Entero largo*	*1*	
WA previous URLs	*Entero largo*	*0*	

XML

Constante	Tipo	Valor	Comentario
Copy XML data source	Entero largo	1	4D conserva una copia del árbol DOM con la imagen, lo cual significa que la imagen puede guardarse en un campo de imagen de la base de datos y luego mostrarse o exportarse en cualquier momento.
DOCTYPE Name	Entero largo	3	Nombre del elemento raíz tal como se definió en el marcador DOCTYPE
Document URI	Entero largo	6	URI de la DTD
Encoding	Entero largo	4	Codificación utilizada (UTF-8, ISO...)
Get XML data source	Entero largo	0	4D sólo lee la fuente de datos XML; no se conserva con la imagen. Esto acelera notablemente la ejecución del comando; sin embargo, como el árbol DOM no se conserva, no es posible guardar o exportar la imagen.
Own XML data source	Entero largo	2	4D exporta el árbol DOM con la imagen. La imagen puede almacenarse o exportarse y la ejecución del comando es rápida. Sin embargo, la referencia XML elementRef puede ser utilizada por otros comandos 4D. Este es el modo por defecto de exportación que se utiliza cuando se omite el parámetro exportType.
PUBLIC ID	Entero largo	1	Identificador público (FPI) de la DTD a la cual el documento es conforme (si la etiqueta DOCTYPE xxx PUBLIC está presente).
SYSTEM ID	Entero largo	2	Identificador sistema
Version	Entero largo	5	Versión de XML aceptada
XML Base64	Entero largo	1	
XML binary encoding	Entero largo	5	Especifica la manera como se convierten los datos binarios. **Valores posibles:** • XML Base64 (valor por defecto): los datos binarios se convierten simplemente en base64 • XML Data URI scheme: los datos binarios se convierten en base64 y se añade el encabezado "data:;base64". Este formato permite principalmente a un navegador decodificar automáticamente una imagen, y también es necesario para insertar imágenes . Para mayor información, consulte http://en.wikipedia.org/wiki/Data_URI_scheme.
XML case insensitive	Entero largo	2	
XML case sensitive	Entero largo	1	
XML CDATA	Entero largo	7	
XML comment	Entero largo	2	
XML convert to PNG	Entero largo	1	
XML DATA	Entero largo	6	
XML data URI scheme	Entero largo	2	
XML date encoding	Entero largo	2	Especifica la forma en que se convierten las fechas 4D. Por ejemplo, !01/01/2003! en la zona horaria de Paris. **Valores posibles:** • XML ISO (valor por defecto): uso del formato xs:datetime sin indicación de la zona horaria. Resultado: "2003-01-01". La parte hora, si está presente en el valor 4D (vía SQL) se pierde. • XML Local: uso del formato xs:date con indicación de zona horaria. Resultado: "2003-01-01 +01:00". La parte hora, si está presente en el valor 4D (vía SQL) se pierde. • XML Datetime local: uso del formato xs:dateTime (ISO 8601). Indicación de la zona horaria. Este formato permite conservar la parte hora, si está presente en el valor 4D (vía SQL). Resultado: "<Date>2003-01-01T00:00:00 +01:00</Date>". • XML UTC: uso del formato xs:date. Resultado: "2003-01-01Z". La parte hora, si está presente en el valor 4D (vía SQL) se pierde. • XML Datetime UTC: uso del formato xs:dateTime (ISO 8601). Este formato permite conservar la parte hora, si está presente en el valor 4D (vía SQL). Resultado: "<Date>2003-01-01T00:00:00Z</Date>".
XML datetime local	Entero largo	3	
XML datetime local absolute	Entero largo	1	
XML datetime UTC	Entero largo	5	

Constante	Tipo	Valor	Comentario
XML disabled	Entero largo	2	
XML DOCTYPE	Entero largo	10	
XML DOM case sensitivity	Entero largo	8	Especifica la sensibilidad a mayúsculas y minúsculas con respecto a los nombres de los elementos de los comandos **DOM Get XML element** y **DOM Count XML elements**. Valores posibles: • XML case sensitive (valor predeterminado): los comandos distinguen entre mayúsculas y minúsculas • XML case insensitive: los comandos no distinguen entre mayúsculas y minúsculas.
XML duration	Entero largo	2	
XML ELEMENT	Entero largo	11	
XML enabled	Entero largo	1	
XML end document	Entero largo	9	
XML end element	Entero largo	5	
XML entity	Entero largo	8	
XML external entity resolution	Entero largo	7	Controla si las entidades externas están definidas en documentos XML. Por razones de seguridad, por defecto, los analizadores XML DOM y SAX de 4D no permiten la resolución de entidades externas. Tenga en cuenta que el alcance de este selector es el proceso de llamada (si es apropiativo) o todos los procesos cooperativos (si se llama desde un proceso cooperativo). Se aplica globalmente a todos los documentos XML (el primer parámetro se ignora, puede pasar una cadena vacía). **Valores posibles:** • XML enabled: permite la resolución de entidades externas en documentos XML • XML disabled (valor predeterminado): no permite la resolución de entidades externas (una declaración de entidad externa genera un error de análisis)
XML indentation	Entero largo	4	Define la indentación del document XML. **Valores posibles:** • XML With indentation (valor por defecto): el documento está indentado. • XML No indentation: el documento no está indentado; su contenido se ubica en una sola línea.
XML ISO	Entero largo	1	
XML local	Entero largo	2	
XML native codec	Entero largo	2	
XML no indentation	Entero largo	2	
XML picture encoding	Entero largo	6	Especifica la forma en la que las imágenes deben convertirse (antes de codificar en base64). **Valores posibles:** • XML Convert to PNG (valor por defecto): las imágenes se convierten en PNG antes de ser codificadas en base64. • XML Native codec: las imágenes se convierten en su primer CODEC nativo de almacenamiento antes de ser codificadas en base64. Debe utilizar estas opciones para codificar imágenes SVG (ver ejemplo del comando **XML SET OPTIONS**).
XML processing instruction	Entero largo	3	
XML raw data	Entero largo	2	
XML seconds	Entero largo	4	
XML start document	Entero largo	1	
XML start element	Entero largo	4	

Constante	Tipo	Valor	Comentario
XML string encoding	*Entero largo*	*1*	*Especifica la forma como las cadenas 4D se convierten en valores de elementos. No concierne a las conversiones en atributos para las cuales XML impone el uso de caracteres de escape.* ***Valores posibles:*** • *XML With escaping (valor por defecto): conversión de las cadenas 4D en valores de elementos XML con reemplazo de caracteres. Los datos de tipo texto son analizados automáticamente de manera que los caracteres prohibidos (<&>') son reemplazados por las entidades XML (&<> '").* • *XML Raw data: las cadenas 4D se envían como datos brutos; 4D no efectúa codificación ni análisis. Los valores 4D se convierten si es posible en fragmentos XML y se insertan como hijo del elemento objetivo. Si un valor no puede considerarse como fragmento XML, se inserta en forma de dato bruto en un nuevo nodo CDATA.*
XML time encoding	*Entero largo*	*3*	*Define la forma como las horas 4D se convierten. Por ejemplo, ?02/00/46? (hora de Paris). La codificación difiere dependiendo de si quiere expresar una hora o una duración.* ***Valores posibles para las horas:*** • *XML Datetime UTC: hora expresada en UTC (Universal Time Coordinated). Note que la conversión a UTC es automática. Resultado: "<Duration>0000-00-00T01:00:46Z</Duration>".* • *XML Datetime local: hora expresada con la diferencia horaria de la máquina del motor de 4D. Resultado: "<Duration>0000-00-00T02:00:46+01:00</Duration>".* • *XML Datetime local absolute (valor por defecto): hora expresada sin indicación de la zona horaria. Sin modificación del valor. Resultado: "<Duration>0000-00-00T02:00:46</Duration>".* ***Valores posibles para las duraciones:*** • *XML Seconds: número de segundos desde la media noche; sin modificación del valor porque expresa una duración. Resultado: "<Duration>7246</Duration>".* • *XML Duration: duración expresada conforme a XML Schema Part 2: Datatypes Second Edition. Sin modificación dle valor ya que expresa una duración. Resultado: "<Duration>PT02H00M46S</Duration>".*
XML UTC	*Entero largo*	*4*	
XML with escaping	*Entero largo*	*1*	
XML with indentation	*Entero largo*	*1*	

Códigos de error

Errores de sintaxis
Errores del analizador de notación objeto (-10718 -> -10701)
Errores de la base de datos
Errores del motor SQL
Errores de red
Errores de gestión de backup
Errores de actualización cliente (-10650 -> -10655)
Errores del Actualizador (-10603 -> -10613)
Errores de gestión de archivos del SO
Errores de gestión de memoria del SO
Errores de gestión de impresión del SO
Errores de gestión de recursos del sistema
Errores de gestión del sonido del sistema
Errores de gestión del puerto serial del sistema
Errores Mac OS
Errores varios (-10700)

Errores de sintaxis

La siguiente tabla lista los códigos y los mensajes de errores de sintaxis que pueden ocurrir durante la ejecución de código en el entorno Diseño o Aplicación. Algunos de estos errores pueden ocurrir en modo interpretado únicamente, algunos sólo en modo compilado y algunos en ambos modos. Puede interceptar estos errores utilizando un método de interrupción de errores instalado por el comando ***ON ERR CALL****.*

Código	Descripción
1	A "(" was expected.
2	A field was expected.
3	The command may be executed only on a field in a subtable.
4	Parameters in the list must all be of the same type.
5	There is no table to which to apply the command.
6	The command may only be executed on a Subtable type field.
7	A Numeric argument was expected.
8	An Alphanumeric argument was expected.
9	The result of a conditional test was expected.
10	The command cannot be applied to this field type.
11	The command cannot be applied between two conditional tests.
12	The command cannot be applied between two Numeric arguments.
13	The command cannot be applied between two Alphanumeric arguments.
14	The command cannot be applied between two Date arguments.
15	The operation is not compatible with the two arguments.
16	The field has no relation.
17	A table was expected.
18	Field types are incompatible.
19	The field is not indexed.
20	An "=" was expected.
21	The method does not exist.
22	The fields must belong to the same table or subtable for a sort or graph.
23	A "<" or ">" was expected.
24	A ";" was expected.
25	There are too many fields for a sort.
26	The field type cannot be Text, Picture, Blob or Subtable.
27	The field must be prefixed by the name of its table.
28	The field type.
29	The value must be 1 or 0.
30	A variable was expected.
31	There is no menu bar with this number.
32	A date was expected.
33	Unimplemented command or function.
34	Accounting files are not open.
35	The sets are from different tables.
36	Invalid table name.
37	A ":=" was expected.
38	This is a function, not a procedure.
39	The set does not exist.
40	This is a procedure, not a function.
41	A variable or field belonging to a subtable was expected.
42	The record cannot be pushed onto the stack.
43	The function cannot be found.
44	The method cannot be found.
45	Field or variable expected.
46	A Numeric or Alphanumeric argument was expected.
47	The field Tipo must be Alphanumeric.
48	Syntax error.
49	This operator cannot be used here.
50	These operators cannot be used together.
51	Module not implemented.
52	An Array was expected.
53	Index out of range.
54	Argument types are incompatible.
55	A Boolean argument was expected.
56	Field, variable, or table expected.
57	An operator was expected.
58	A ")" was expected.
59	This kind of argument was not expected here.
60	A parameter or a local variable cannot be used in an EXECUTE statement in a compiled database.
61	The Tipo of an Array cannot be modified in a compiled database.
62	The command cannot be applied to a subtable.
63	The field is not indexed.
64	A picture field or variable was expected.
65	The value should contain 4 characters.
66	The value should not contain more than 3 characters.

67	*This command cannot be executed on 4D Server.*
68	*A list was expected.*
69	*An external window reference was expected.*
70	*The command cannot be applied between two Picture arguments.*
71	*The SET PRINT MARKER command can only be called in the header of a form being printed.*
72	*A pointer Array was expected.*
73	*A numeric Array was expected.*
74	*The size of arrays does not match.*
75	*No pointer on local arrays.*
76	*Bad Array Type.*
77	*Bad variable name.*
78	*Invalid sort paramater.*
79	*This command cannot be executed during the draw of a list.*
80	*Too many query arguments.*
81	*The form was not found.*

Tips

Algunos códigos de error señalan errores de sintaxis por errores de digitación. Por ejemplo, usted obtiene el error #37 si ejecuta la expresión v=0 cuando en realidad quería escribir v:=0. Puede eliminar el error corrigiendo su código en el editor de métodos.

*Algunos de estos códigos de error señalan errores de programación simples. Por ejemplo, usted obtiene el error #5 si ejecuta un comando **ADD RECORD,** sin indicar el nombre de la tabla en el parámetro correspondiente, y haber definido la tabla por defecto con la ayuda del comando **DEFAULT TABLE**. En este caso, se corrige el error definiendo una tabla por defecto o pasando un nombre de tabla en el parámetro correspondiente.*

*Algunos de estos códigos de error señalan errores relacionados con la estructura de la base. Por ejemplo, usted obtiene el error #16 si aplica el comando **RELATE ONE** a un campo que no está relacionado con otro campo. Elimine el error modificando su código o creando una relación a partir del campo.*

Ciertos errores no siempre están en el lugar donde su código se detuvo. Por ejemplo, si en una subrutina usted obtiene el error #53 (index out of range) en la línea vpCamp:=Campo($1;$2), el error es porque se pasaron a la subrutina como parámetros números incorrectos de tablas o de campos. Por lo tanto, el error está ubicado en el método llamante y no donde ocurre el error. En este caso, haga seguimiento a su código en la ventana del depurador para determinar qué línea de código contiene el error, para corregirlo en el editor de métodos.

Errores del analizador de notación objeto (-10718 -> -10701)

La tabla siguiente lista los códigos y los mensajes de error que pueden ocurrir al analizar el código 4D usando la notación objeto:

Código	***Descripción***
-10718	*El valor 'null' no es soportado por este tipo de expresión*
-10717	*No se puede crear un puntero en una propiedad de objeto*
-10716	*Objeto o colección esperada*
-10715	*Propiedad desconocida*
-10714	*Índice o propiedad no válida*
-10713	*El índice de la colección está fuera de los límites*
-10712	*El índice debe ser un número finito y positivo*
-10711	*No se puede cambiar una propiedad en una colección*
-10710	*No se puede eliminar una propiedad de una colección*
-10709	*No se puede agregar la propiedad a la colección*
-10708	*Colección indefinida*
-10707	*Tipo de propiedad no válido*
-10706	*No se puede acceder a las propiedades del objeto por índice*
-10705	*Intento de agregar una propiedad a un objeto no escalable*
-10704	*Intento de eliminar una propiedad de objeto no configurable*
-10703	*Intento de cambiar una propiedad de objeto no configurable*
-10702	*Intento de escribir en una propiedad de objeto de sólo lectura*
-10701	*Objeto no definido*

Errores de la base de datos

Esta tabla lista los códigos de error generados por el motor de base de datos de 4D. Estos errores de bajo nivel pueden ocurrir durante operaciones relacionadas con el motor de la base, tal como interrupciones del usuario, errores de privilegios y objetos dañados.

Código	Descripción
-10602	Can't execute the given command because there is no open print job.
-10601	The **OBJECT DUPLICATE** command does not work during printing operations.
-10600	This BLOB could not be read. It may be damaged.
-10532	Cannot call error handling project method 'methodName.
-10526	Can't create database {database_name}.
-10525	Can't delete existing database {database_name}.
-10524	Can't open database {database_name}: current data file not found at last known location: '{datafile_path}'.
-10523	Can't open database {database_name}: current data file is not defined.
-10522	Can't load component {database_name} because it's not in Unicode mode.
-10521	Can't open database {database_name} because it's not in Unicode mode.
-10520	Unauthorized user: Only an Administrator or Designer can execute this command.
-10519	Bad url: {url}
-10518	Assert failed: {assertion}
-10517	Failed to synchronize {folder_name} folder.
-10516	The server is running a maintenance operation, please reconnect later.
-10515	Your attempt to connect to 4D Server has been denied
-10514	The maximum number of concurrent users for your license has been reached
-10513	Can't call {command_name} command from a remote 4D Developer.
-10512	The encoding is not supported
-10511	Can't call command "{command_name}" from a component.
-10510	Can't load component "{component_name}".
-10509	Can't open database "{database_name}".
-10508	Project method not found.
-10507	This version does not allow a compiled database to be opened.
-10506	Limit of the Standard Edition version.
-10505	Client and server have version numbers that are incompatible.
-10504	Index page # is out of range (index needs to be repaired).
-10503	Record # is out of range (during **GOTO RECORD**, or data file needs to be repaired). (see note 3)
-10502	Invalid record structure (data file needs to be repaired).
-10501	Invalid index page (index needs to be repaired).
-10500	Invalid record address (database needs to be repaired).
-9999	No more room to save the record. (see note 4)
-9998	Duplicated key.
-9997	Maximum number of records has been reached.
-9996	Stack is full (too much recursion or nested calls).
-9995	Demo limit has been reached.
-9994	Serial communication interrupted by the user—user pressed Ctrl-Alt-Shift (Windows) or Command-Option-Shift (Mac OS).
-9993	Menu bar is damaged (database needs to be repaired).
-9992	Wrong password.
-9991	Privileges error.
-9990	Time-out error.
-9989	Invalid structure (database needs to be repaired).
-9988	The form cannot be loaded. Either the form or the structure is damaged.
-9987	Some other records are already related to this record.
-9986	Record locked during an automatic deletion action.
-9985	Recursive integrity.
-9984	Transaction has been cancelled because of a duplicated index key error.
-9983	The same external package is installed twice.
-9982	The record was not loaded because it is not in the selection on the workstation.
-9981	Invalid field name/field number definition table sent by the workstation.
-9980	The file cannot be created because the structure is locked.
-9979	Unknown user.
-9978	Bad user password.
-9977	The selection does not exist.
-9976	Backup in progress; no modification allowed.
-9975	Transaction index page could not be loaded.
-9974	Record has already been deleted.
-9973	The TRIC resources are not the same.
-9972	Table number is out of range requested by workstation.
-9971	Field number is out of range requested by workstation.
-9970	Field is not indexed.
-9969	Invalid field type requested by workstation.
-9968	Invalid selected record number requested by workstation.
-9967	The record could not be modified because it could not be loaded.
-9966	Invalid type requested by a workstation.
-9965	Bad search definition table sent by a workstation.

-9964 Bad sort definition table sent by a workstation.
-9963 Invalid record number requested by a workstation.
-9962 The backup cannot be run because the server is shutting down.
-9961 The backup process is not currently running.
-9960 4D Backup is not installed on the server.
-9959 The backup process has already been started by another user or process.
-9958 The process could not be started.
-9957 The choice list is locked.
-9956 Versions of 4D Client and 4D Server are different.
-9955 QuickTime is not installed.
-9954 There is no current record.
-9953 There is no Log file.
-9952 Invalid data segment header.
-9951 This field has no relation.
-9950 Invalid data segment number.
-9949 License or privilege error.
-9948 A modal dialog is activated.
-9947 The “Allow 4D Open connections” check box has not been selected.
-9946 Unable to clear the named selection because it does not exist.
-9945 CD-ROM 4D Runtime error; writing operations are not allowed.
-9944 The user does not belong to the 4D Open access group.
-9943 4D Connectivity Plug-ins version error.
-9942 4D Client licensing scheme is incompatible with this version of 4D Server.
-9941 Unknown EX_GESTALT selector.
-9940 4D Extension initialization failed.
-9939 External routine not found.
-9938 The current record has been changed from within the trigger.
-9937 Password System is locked by another user.
-9936 External password code does not match to the database one
-9935 The XML file is not valid or is not well-formed.
-9934 The XML file is not well-formed.
-9933 The XML file is not valid.
-9932 The XML DLL is not loaded.
-9931 The index for this attribute is invalid.
-9930 There is no attribute with this name for this element.
-9929 The index for this element is invalid.
-9928 The name of the element is unknown.
-9927 The referenced element is not the “root”.
-9926 The referenced element is invalid.
-9925 The referenced element is null.
-9924 The file must be opened in read only
-9923 The attribute name is not valid
-9922 Missing the parameter value in the attributes definition
-9921 Attempt to write a XML Prolog on a non-empty document
-9920 The type of the node is invalid
-9919 This encoding is not supported.
-9918 The name of the element is invalid.
-9917 The type of the array passed in parameter is invalid.
-9916 The element is not open.
-9915 The document's reference is invalid.
-9914 Internal fault.
-9913 Network fault.
-9912 HTTP fault.
-9911 Parser fault.
-9910 Soap fault.
-9909 No window available to run the form.
-9855 Invalid parameter number 5.
-9854 Invalid parameter number 4.
-9853 Invalid parameter number 3.
-9852 Invalid parameter number 2.
-9851 Invalid parameter number 1.
-9850 Invalid area parameter passed to an external command.
-9803 Unnamed object found in the form "{form}".
-9802 Object path not unique: {path}. Verify the application with the MSC.
-9801 Unable to open the method: {path}
-9800 One of the processes modified the access rights.
-9799 Error during preview initialization.
-9778 An error occurred while loading the dictionary

-9777	*The method language does not match the application localization: {path}*
-9776	*Unable to create method: {path}*
-9775	*Unable to write method code: {path}*
-9774	*Unable to read method code: {path}*
-9773	*Unable to write method comments: {path}*
-9772	*Unable to read method comments: {path}*
-9771	*Unable to write method properties: {path}*
-9770	*Unable to read method properties: {path*
-9769	*Invalid method property: {path}*
-9768	*Invalid object path: {path}*
-9767	*Cannot update methods. One or more resources are locked.*
-9766	*The method is currently being edited.*
-9765	*One or more comments are currently being edited.*
-9764	*The comment is currently being edited.*
-9763	*The command cannot be executed in a component.*
-9762	*The command cannot be executed in a compiled database.*
-9761	*Invalid object type.*
-9760	*It is not possible to move or copy the XML node to the specified position.*
-9759	*The Object Library could not be opened.*
-9758	*The user form already exists.*
-9757	*The user form does not exist.*
-9756	*There is no user structure file.*
-9755	*The user form does not have a name.*
-9754	*This command cannot be used from a dialog window.*
-9753	*The source form does not exist.*
-9752	*The user form cannot be created.*
-9751	*The source form is not accessible by the user.*
-9750	*The source form is not editable.*
-1	*Invalid table number requested by a Plug-In*
1001	*No column to import in table {TableName}*
1002	*Wrong table to import*
1003	*No column to export in table {TableName}*
1004	*Wrong table to export*
1006	*Program interrupted by user—user pressed Alt-click (Windows) or Option-click (Mac OS)*
1006	*Cannot save structure of database {BaseName}*
1007	*Cannot create data file of database: {BaseName}*
1008	*Wrong data segment in database {BaseName}*
1009	*Memory is full*
1010	*Cannot load a table definition of database {BaseName}*
1011	*Cannot open data file of database {BaseName}*
1012	*Data segment is full in database: {BaseName}*
1013	*Cannot save data segment in database {BaseName}*
1014	*Cannot read data segment of database {BaseName}*
1015	*Wrong database header in database {BaseName}*
1016	*Cannot create table in database {BaseName}*
1017	*Cannot read index list of database {BaseName}*
1018	*Cannot write index list of database {BaseName}*
1019	*Wrong table reference in database {BaseName}*
1020	*Wrong field reference in database {BaseName}*
1021	*Invalid index type in database {BaseName}*
1022	*Invalid field name in table {TableName} of database {BaseName}*
1023	*Invalid database name*
1024	*Cannot open structure of database {BaseName}*
1025	*Cannot create structure of database {BaseName}*
1026	*Cannot load bit selection of database {BaseName}*
1027	*Cannot load set of database {BaseName}*
1028	*Cannot modify set of database {BaseName}*
1029	*Cannot save set of database {BaseName}*
1030	*Cannot save BLOB {BlobNum} in table {TableName} of database {BaseName}*
1031	*Cannot load BLOB {BlobNum} in table {TableName} of database {BaseName}*
1032	*Cannot allocate BLOB {BlobNum} in table {TableName} of database {BaseName}*
1033	*Cannot load data bit table of database {BaseName}*
1034	*Cannot save data bit table of database {BaseName}*
1035	*Cannot load table of data bit tables of database {BaseName}*
1036	*Cannot save table of data bit tables in database {BaseName}*
1037	*Cannot close data segment of database {BaseName}*
1038	*Cannot delete data segment of database {BaseName}*
1039	*Cannot open data segment of database {BaseName}*

1040 Cannot create data segment of database {BaseName}
1041 Cannot allocate space in data segment
1042 Cannot free space in data segment of database {BaseName}
1043 File is write protected
1044 Cannot access field in record {RecNum} in table {TableName} of database {BaseName}
1045 Field definition code is missing in record {RecNum} in table {TableName} of database {BaseName}
1046 Cannot save record {RecNum} in table {TableName} of database {BaseName}
1047 Cannot load record {RecNum} in table {TableName} of database {BaseName}
1048 Cannot allocate record in table {TableName} of database {BaseName}
1049 Cannot update record {RecNum} in table {TableName} of database {BaseName}
1050 Wrong header for record {RecNum} in table {TableName} of database {BaseName}
1051 Cannot save definition of table {TableName} of database {BaseName}
1052 Cannot update definition of table {TableName} of database {BaseName}
1053 Field name already exists
1055 Cannot update index values while saving a record in table {TableName} of database {BaseName}
1056 Cannot update BLOBs while saving a record in table {TableName} of database {BaseName}
1057 Cannot delete BLOBs while saving or deleting a record in table {TableName} of database {BaseName}
1058 Cannot add field in table {TableName} of database {BaseName}
1059 Cannot allocate table in memory
1061 Cannot allocate record in memory
1062 Cannot completely delete table {TableName} of database {BaseName}
1063 Cannot lock record {RecNum} in table {TableName} of database {BaseName}
1064 Cannot unlock record {RecNum} in table {TableName} of database {BaseName}
1065 Cannot delete record {RecNum} in table {TableName} of database {BaseName}
1066 Record {RecNum} is locked in table {TableName} of database {BaseName}
1067 Sequential search could not be completed in table {TableName} of database {BaseName}<
1068 Cannot save header for table {TableName} of database {BaseName}<
1069 Cannot import data in table {TableName} of database {BaseName}
1070 Cannot load index header of database {BaseName}
1071 Cannot save header for index {IndexName} of database {BaseName}
1072 Cannot get page address for index {IndexName} of database {BaseName}
1073 Cannot set page address for index {IndexName} of database {BaseName}
1074 Cannot drop index {IndexName} of database {BaseName}
1075 Cannot sort index {IndexName} of database {BaseName}
1076 Cannot load page for index {IndexName} of database {BaseName}
1077 Cannot save page for index {IndexName} of database {BaseName}
1078 Cannot insert key into index {IndexName} of database {BaseName}
1079 Cannot delete key from index {IndexName} of database {BaseName}
1080 Cannot completely delete index {IndexName} of database {BaseName}
1081 Cannot complete scan on index {IndexName} of database {BaseName}
1082 Cannot complete sort of index {IndexName} of database {BaseName}
1083 Cannot insert key into page of index {IndexName} of database {BaseName}
1084 Cannot delete key from page of index {IndexName} of database {BaseName}
1085 Cannot load index cluster of database {BaseName}
1086 Cannot add to index cluster of database {BaseName}
1087 Cannot delete from index cluster of database {BaseName}
1088 Index {IndexName} is invalid
1089 SQL syntax error (Obsolete)
1090 SQL token not found (Obsolete)
1091 Not implemented
1092 Cannot register code
1093 Operation in progress cancelled by user
1094 Transaction conflict
1095 Invalid table name in database {BaseName}
1096 Table {TableName} is locked in database {BaseName}
1097 Database {BaseName} is locked
1098 Data address is invalid in database {BaseName}
1099 Record is empty
1100 Wrong source field
1101 Wrong destination field
1102 Invalid relation name
1103 Field types do not match
1104 Relation is empty<
1105 Cannot load relations list from database {BaseName}
1106 Cannot save relations list into database {BaseName}
1107 Query and lock not completed; at least one record locked somewhere else
1108 Invalid record
1109 Wrong record ID

1110 *Relation already exists in database {BaseName}*
1111 *Index already exists in database {BaseName}*
1112 *Wrong comparison operator*
1113 *End of data buffer*
1114 *Wrong DB4D version number*
1115 *Duplicated key*
1116 *Mandatory field is Null in record {RecNum} in table {TableName}*
1117 *Cannot set field to mandatory for table {TableName}*
1118 *Cannot get exclusive access to table {TableName}*
1119 *Cannot check referential integrity of table {TableName}*
1120 *Reference integrity: some foreign keys still exist matching primary key on record {RecNum} in table {TableName}*
1121 *Cannot delete all records in selection of table {TableName}*
1122 *BLOB is Null*
1123 *Invalid database context*
1124 *Invalid relation reference*
1125 *Invalid record name in table {TableName}*
1126 *Wrong field type*
1127 *Cannot load extra properties*
1128 *Cannot save extra properties*
1129 *Subrecord ID is out of range*
1130 *Duplicate name for index {IndexName} in database {BaseName}*
1131 *Invalid name for index {IndexName} in database {BaseName}*
1132 *Wrong key value on index {IndexName}*
1133 *Wrong type for index {IndexName}*
1134 *Invalid accessor*
1135 *Accessor is read only*
1136 *Null value not accepted*
1137 *THIS is Null*
1138 *Selection is Null*
1139 *Database {BaseName} is write protected*
1140 *Database {BaseName} is closing*
1141 *Invalid transaction*
1142 *Array limit is exceeded*
1143 *Creator of array values is missing*
1144 *Cannot build selection of table {TableName}*
1145 *Busy object*
1146 *Data file does not match structure file*
1147 *Cannot start listener*
1148 *Cannot start server*
1149 *No listener*
1150 *Task is dying*
1151 *Invalid request tag*
1152 *Invalid context ID*
1153 *Not enough temporary space on disk*
1154 *Data set is Null*
1155 *No primary key matching the foreign key*
1156 *Type of field {FieldName} of table {TableName} does not support being set to Unique*
1157 *Type of field {FieldName} of table {TableName} does not support being set to NEVER NULL*
1158 *Cannot alter primary key definition of table {TableName}*
1159 *Maximum number of records has been reached for table {TableName}*
1160 *Maximum number of BLOBs has been reached for table {TableName}*
1161 *Indice is out of range*
1162 *Query is invalid*
1163 *Record is NULL*
1164 *Object is NULL*
1165 *Wrong owner for this object*
1166 *Object was not locked*
1167 *Object is locked by another context*
1168 *Internal error on remote connection*
1169 *Invalid table number*
1170 *Invalid field number*
1171 *Invalid Database ID*
1172 *Invalid Parameter*
1173 *Cache Flushing did not complete*
1174 *Data Flushing did not complete*
1175 *Structure Flushing did not complete*
1176 *Log file is invalid for database {BaseName}*
1177 *Log file cannot be found for database {BaseName}*

1178 *Last operation in log file doest not match the one in database {BaseName}*
1179 *Log file does not match database {BaseName}*
1180 *Data Table has been deleted*
1181 *Keys are not unique in index {IndexName}*
1182 *Cannot create log file for database {DataBase}*
1183 *Cannot write into log file of database {DataBase}*
1184 *Cannot drop table {TableName} in database {DataBase}*
1185 *Remote database cannot be opened<*
1186 *Log file cannot be integrated into database {DataBase}*
1187 *Internal Computation on Set cannot be completed*
1188 *Array cannot be saved*
1189 *Array cannot be loaded*
1190 *Sequence number header cannot be loaded*
1191 *Cannot select record*
1192 *Record cannot be created*
1193 *Cannot complete selection to array in table {TableName} of database {BaseName}*
1194 *Cannot complete array to selection in table {TableName} of database {BaseName}*
1195 *Cannot complete sequential sort*
1196 *Selection cannot be locked*
1197 *Index key cannot be loaded*
1198 *Index key cannot be saved*
1199 *Index key cannot be built*
1200 *Query cannot be completed*
1201 *Query cannot be analyzed*
1202 *Formula could not be processed on this column*
1203 *Query could not be completed*
1204 *Query could not be analyzed*
1205 *Could not get all distinct values of table {TableName} of database {BaseName}*
1206 *Cannot build array of values in table {TableName} of database {BaseName}*
1207 *Selection cannot be loaded*
1208 *Cannot send data*
1209 *Cannot receive request reply*
1210 *Cannot send request*
1211 *Cannot create connection*
1212 *Index {IndexName} cannot be quickly created in database {BaseName}*
1213 *Cannot build distinct index keys*
1214 *Selection cannot be sorted in table {TableName} of database {BaseName}*
1215 *Address table cannot loaded*
1216 *Address table cannot modified*
1217 *Cannot allocate new entry into address table*
1218 *Cannot allocate free entry from address table*
1219 *Transaction temporary record cannot be saved*
1220 *Transaction temporary blob cannot be saved*
1221 *Transaction temporary record cannot be loaded*
1222 *Transaction temporary blob cannot be loaded*
1223 *Transaction cannot be started*
1224 *Transaction cannot be commited*
1225 *Cannot get extra property*
1226 *Cannot set extra property*
1227 *Table name is already used*
1228 *Cannot get list of NULL keys from index {IndexName}*
1229 *Cannot modify list of NULL keys of index {IndexName}*
1230 *Invalid key for index {IndexName}*
1231 *Cannot set log file*
1232 *Context is NULL*
1233 *Database {BaseName} cannot be locked*
1234 *Wrong field reference in record# {RecNum} of table: {TableName} of database: {BaseName}*
1235 *Wrong field reference in table: {TableName} of database: {BaseName}*
1236 *Cannot read data from temporary transaction file*
1237 *Cartesian Product failed*
1238 *Cannot merge selections*
1239 *The format of the {BaseName} database cannot be upgraded in read only mode*
1240 *Wrong Header*
1241 *Wrong CheckSum*
1243 *Cannot load data tables of database: {BaseName}*
1244 *The list of foreign key constraints is not empty for table: {TableName} of database: {BaseName}*
1245 *Address entry is not empty*
1246 *Cannot pre-allocate address*

1247	*Cannot update new record in table {TableName} of database {BaseName}*
1248	*Cannot save new record in table {TableName} of database {BaseName}*
1249	*Cannot save subrecord*
1250	*Cannot save record*
1251	*Cannot lock structure object definition*
1252	*Cannot unlock structure object definition*
1253	*Invalid relation number*
1254	*Circular Reference in records address table of table {TableName} of database {BaseName}*
1255	*Circular Reference in blobs address table of table {TableName} of database {BaseName}*
1256	*Duplicated Schema Name in database {BaseName}*
1257	*Schema cannot be saved in database {BaseName}*
1258	*Schema cannot be deleted in database {BaseName}*
1259	*Schema cannot be renamed in database {BaseName}*
1260	*Selected log file is too recent for database {BaseName}*
1261	*Selected log file is too old for database {BaseName}*
1262	*Some DataTables do not have matching table definitions for database {BaseName}*
1263	*Given stamp does not match current one for record# {RecNum} of table {TableName}*
1264	*A primary key is needed and missing in table {TableName}*
1265	*Invalid field*
1266	*Unknown field type (this version of 4D is perhaps too old)*
1267	*Stack Overflow*
1268	*DataTable cannot be rebuilt for database {BaseName}*
1269	*DataTable cannot be found*
1270	*Missing structure could not be rebuilt*
1271	*Invalid REST request handler*
1272	*Some records are still locked in table {tableName} in database {BaseName}*
1273	*Cannot drop table {TableName} in database {BaseName}*
1274	*The current journal file of database {BaseName} is not available. For data security reasons, write operations are stopped. Please contact your administrator as soon as possible.*
1275	*"(" is missing at the beginning of the Javascript statement in this query.*
1276	*Datastream has an invalid header.*
1277	*The database journal integration failed at entry# {p1}.*
1278	*Javascript code not allowed in the query.*
1300	*Invalid Selection Type*
1301	*Array is too big*
1302	*Array size does not match*
1303	*Invalid Selection ID*
1304	*Invalid Selection Part*
4001	*Invalid table number requested by a Plug-In*
4002	*Invalid record number requested by a Plug-In*
4003	*Invalid field number requested by a Plug-In*
4004	*Access to a table's current record requested by a Plug-in while there is no current record*

Notas

1. *Mientras algunos de los errores listados reflejan problemas serios, por ejemplo, -10502 Invalid record structure (data file needs to be repaired), otros errores pueden presentarse con frecuencia y ser tratados por un método de proyecto **ON ERR CALL**. Por ejemplo, es común interceptar el error –9998 Duplicated key si su aplicación tiene la posibilidad de crear valores idénticos para una tabla que contiene un campo indexado que tiene la propiedad único.*
2. *Algunos de los errores listados nunca se presentan a nivel del lenguaje de 4D. Pueden ocurrir y ser tratados por las rutinas del motor de la base o cuando se utiliza, por ejemplo, 4D Open.*
3. *El error -10503 Record # is out of range significa por lo general que su código (por ejemplo el comando **GOTO RECORD**) intenta acceder a un registro que no existe actualmente (o nunca ha existido). En ciertos casos, más inusuales, este error puede significar que la base debe ser reparada.*
4. *El error -9999 No more room to save the record ocurre cuando el archivo de datos de su base está lleno o ubicado en un volumen lleno. Este error también puede ser generado si el archivo de datos está bloqueado o almacenado en un volumen bloqueado.*

Errores del motor SQL

*El motor SQL de 4D devuelve errores específicos los cuales se listan a continuación. Estos errores pueden ser interceptados utilizando un método de gestión de errores instalado por el comando **ON ERR CALL** y analizado utilizando el comando **GET LAST SQL ERROR.***

Errores genéricos

1001	*INVALID ARGUMENT*
1002	*INVALID INTERNAL STATE*
1003	*SQL SERVER IS NOT RUNNING*
1004	*Access denied*
1005	*FAILED TO LOCK SYNCHRONIZATION PRIMITIVE*
1006	*FAILED TO UNLOCK SYNCHRONIZATION PRIMITIVE*
1007	*SQL SERVER IS NOT AVAILABLE*
1008	*COMPONENT BRIDGE IS NOT AVAILABLE*
1009	*REMOTE SQL SERVER IS NOT AVAILABLE*
1010	*EXECUTION INTERRUPTED BY USER*

Errores semánticos

1101 Table '{key1}' does not exist in the database.
1102 Column '{key1}' does not exist.
1103 Table '{key1}' is not declared in the FROM clause.
1104 Column name reference '{key1}' is ambiguous.
1105 Table alias '{key1}' is the same as table name.
1106 Duplicate table alias - '{key1}'.
1107 Duplicate table in the FROM clause - '{key1}'.
1108 Operation {key1} {key2} {key3} is not type safe.
1109 Invalid ORDER BY index - {key1}.
1110 Function {key1} expects one parameter, not {key2}.
1111 Parameter {key1} of type {key2} in function call {key3} is not implicitly convertible to {key4}.
1112 Unknown function - {key1}.
1113 Division by zero.
1114 Sorting by indexed item in the SELECT list is not allowed - ORDER BY item {key1}.
1115 DISTINCT NOT ALLOWED
1116 Nested aggregate functions are not allowed in the aggregate function {key1}.
1117 Column function is not allowed.
1118 Cannot mix column and scalar operations.
1119 Invalid GROUP BY index - {key1}.
1120 GROUP BY index is not allowed.
*1121 GROUP BY is not allowed with 'SELECT * FROM ...'.*
1122 HAVING is not an aggregate expression.
1123 Column '{key1}' is not a grouping column and cannot be used in the ORDER BY clause.
1124 Cannot mix {key1} and {key2} types in the IN predicate.
1125 Escape sequence '{key1}' in the LIKE predicate is too long. It must be exactly one character.
1126 Bad escape character - '{key1}'.
1127 Unknown escape sequence - '{key1}'.
1128 Column references from more than one query in aggregate function {key1} are not allowed.
1129 Scalar item in the SELECT list is not allowed when GROUP BY clause is present.
1130 Sub-query produces more than one column.
1131 Subquery must return one row at the most but instead it returns {key1}.
1132 INSERT value count {key1} does not match column count {key2}.
1133 Duplicate column reference in the INSERT list - '{key1}'.
1134 Column '{key1}' does not allow NULL values.
1135 Duplicate column reference in the UPDATE list - '{key1}'.
1136 Table '{key1}' already exists.
1137 Duplicate column '{key1}' in the CREATE TABLE command.
1138 DUPLICATE COLUMN IN COLUMN LIST
1139 More than one primary key is not allowed.
1140 Ambiguous foreign key name - '{key1}'.
1141 Column count {key1} in the child table does not match column count {key2} in the parent table of the foreign key definition.
1142 Column type mismatch in the foreign key definition. Cannot relate {key1} in child table to {key2} in parent table.
1143 Failed to find matching column in parent table for '{key1}' column in child table.
1144 UPDATE and DELETE constraints must be the same.
1145 FOREIGN KEY DOES NOT EXIST
1146 Invalid LIMIT value in SELECT command - {key1}.
1147 Invalid OFFSET value in SELECT command - {key1}.
1148 Primary key does not exist in table '{key1}'.
1149 FAILED TO CREATE FOREIGN KEY
1150 Column '{key1}' is not part of a primary key.
1151 FIELD IS NOT UPDATEABLE
1152 FOUND VIEW COLUMN
1153 Bad data type length '{key1}'.
1154 EXPECTED EXECUTE IMMEDIATE COMMAND
1155 INDEX ALREADY EXISTS
1156 Auto-increment option is not allowed for column '{key1}' of type {key2}.
1157 SCHEMA ALREADY EXISTS
1158 SCHEMA DOES NOT EXIST
1159 Cannot drop system schema
1160 CHARACTER ENCODING NOT ALLOWED

Errores de implementación

1203 FUNCTIONALITY IS NOT IMPLEMENTED
1204 Failed to create record {key1}.
1205 Failed to update field '{key1}'.
1206 Failed to delete record '{key1}'.
1207 NO MORE JOIN SEEDS POSSIBLE
1208 FAILED TO CREATE TABLE
1209 FAILED TO DROP TABLE
1210 CANT BUILD BTREE FOR ZERO RECORDS
1211 COMMAND COUNT GREATER THAN ALLOWED
1212 FAILED TO CREATE DATABASE
1213 FAILED TO DROP COLUMN
1214 VALUE IS OUT OF BOUNDS
1215 FAILED TO STOP SQL_SERVER
1216 FAILED TO LOCALIZE
1217 Failed to lock table for reading.
1218 FAILED TO LOCK TABLE FOR WRITING
1219 TABLE STRUCTURE STAMP CHANGED
1220 FAILED TO LOAD RECORD
1221 FAILED TO LOCK RECORD FOR WRITING
1222 FAILED TO PUT SQL LOCK ON A TABLE
1223 FAILED TO RETAIN COOPERATIVE TASK
1224 FAILED TO LOAD INFILE

Error de análisis

1301 PARSING FAILED

Errores de acceso al lenguaje Runtime

1401 COMMAND NOT SPECIFIED
1402 ALREADY LOGGED IN
1403 SESSION DOES NOT EXIST
1404 UNKNOWN BIND ENTITY
1405 INCOMPATIBLE BIND ENTITIES
1406 REQUEST RESULT NOT AVAILABLE
1407 BINDING LOAD FAILED
1408 COULD NOT RECOVER FROM PREVIOUS ERRORS
1409 NO OPEN STATEMENT
1410 RESULT EOF
1411 BOUND VALUE IS NULL
1412 STATEMENT ALREADY OPENED
1413 FAILED TO GET PARAMETER VALUE
1414 INCOMPATIBLE PARAMETER ENTITIES
1415 Query parameter is either not allowed or was not provided.
1416 COLUMN REFERENCE PARAMETERS FROM DIFFERENT TABLES
1417 EMPTY STATEMENT
1418 FAILED TO UPDATE VARIABLE
1419 FAILED TO GET TABLE REFERENCE
1420 FAILED TO GET TABLE CONTEXT
1421 COLUMNS NOT ALLOWED
1422 INVALID COMMAND COUNT
1423 INTO CLAUSE NOT ALLOWED
1424 EXECUTE IMMEDIATE NOT ALLOWED
1425 ARRAY NOT ALLOWED IN EXECUTE IMMEDIATE
1426 COLUMN NOT ALLOWED IN EXECUTE IMMEDIATE
1427 NESTED BEGIN END SQL NOT ALLOWED
1428 RESULT IS NOT A SELECTION
1429 INTO ITEM IS NOT A VARIABLE
1430 VARIABLE WAS NOT FOUND
1431 PTR OF PTR NOT ALLOWED
1432 POINTER OF UNKNOWN TYPE

Errores de formato de fecha

1501 SEPARATOR_EXPECTED
1502 FAILED TO PARSE DAY OF MONTH
1503 FAILED TO PARSE MONTH
1504 FAILED TO PARSE YEAR
1505 FAILED TO PARSE HOUR
1506 FAILED TO PARSE MINUTE
1507 FAILED TO PARSE SECOND
1508 FAILED TO PARSE MILLISECOND
1509 INVALID AM PM USAGE
1510 FAILED TO PARSE TIME ZONE
1511 UNEXPECTED CHARACTER
1512 Failed to parse timestamp.
1513 Failed to parse duration.
1551 FAILED TO PARSE DATE FORMAT

Errores Lexer

1601 NULL INPUT STRING
1602 NON TERMINATED STRING
1603 NON TERMINATED COMMENT
1604 INVALID NUMBER
1605 UNKNOWN START OF TOKEN
1606 NON TERMINATED NAME/ closing ']' is missing*
1607 NO VALID TOKENS

Errores de validación para errores de pila - errores de estado que siguen a errores directos

1701 Failed to validate table '{key1}'.
1702 Failed to validate FROM clause.
1703 Failed to validate GROUP BY clause.
1704 Failed to validate SELECT list.
1705 Failed to validate WHERE clause.
1706 Failed to validate ORDER BY clause.
1707 Failed to validate HAVING clause.
1708 Failed to validate COMPARISON predicate.
1709 Failed to validate BETWEEN predicate.
1710 Failed to validate IN predicate.
1711 Failed to validate LIKE predicate.
1712 Failed to validate ALL ANY predicate.
1713 Failed to validate EXISTS predicate.
1714 Failed to validate IS NULL predicate.
1715 Failed to validate subquery.
1716 Failed to validate SELECT item {key1}.
1717 Failed to validate column '{key1}'.
1718 Failed to validate function '{key1}'.
1719 Failed to validate CASE expression.
1720 Failed to validate command parameter.
1721 Failed to validate function parameter {key1}.
1722 Failed to validate INSERT item {key1}.
1723 Failed to validate UPDATE item {key1}.
1724 Failed to validate column list.
1725 Failed to validate foreign key.
1726 Failed to validate SELECT command.
1727 Failed to validate INSERT command.
1728 Failed to validate DELETE command.
1729 Failed to validate UPDATE command.
1730 Failed to validate CREATE TABLE command.
1731 Failed to validate DROP TABLE command.
1732 Failed to validate ALTER TABLE command.
1733 Failed to validate CREATE INDEX command.
1734 Failed to validate LOCK TABLE command.
1735 Failed to calculate LIKE predicate pattern.

Errores de ejecución para errores de pila - errores de estado que siguien los errores directos

1801 Failed to execute SELECT command.
1802 Failed to execute INSERT command.
1803 Failed to execute DELETE command.
1804 Failed to execute UPDATE command.
1805 Failed to execute CREATE TABLE command.
1806 Failed to execute DROP TABLE command.
1807 Failed to execute CREATE DATABASE command.
1808 Failed to execute ALTER TABLE command.
1809 Failed to execute CREATE INDEX command.
1810 Failed to execute DROP INDEX command.
1811 Failed to execute LOCK TABLE command.
1812 Failed to execute TRANSACTION command.
1813 Failed to execute WHERE clause.
1814 Failed to execute GROUP BY clause.
1815 Failed to execute HAVING clause.
1816 Failed to aggregate.
1817 Failed to execute DISTINCT.
1818 Failed to execute ORDER BY clause.
1819 Failed to build DB4D query.
1820 Failed to calculate comparison predicate.
1821 Failed to execute subquery.
1822 Failed to calculate BETWEEN predicate.
1823 Failed to calculate IN predicate.
1824 Failed to calculate ALL/ANY predicate.
1825 Failed to calculate LIKE predicate.
1826 Failed to calculate EXISTS predicate.
1827 Failed to calculate NULL predicate.
1828 Failed to perform arithmetic operation.
1829 Failed to calculate function call '{key1}'.
1830 Failed to calculate case expression.
1831 Failed to calculate function parameter '{key1}'.
1832 Failed to calculate 4D function call.
1833 Failed to sort while executing ORDER BY clause.
1834 Failed to calculate record hash.
1835 Failed to compare records.
1836 Failed to calculate INSERT value {key1}.
1837 DB4D QUERY FAILED
1838 FAILED TO EXECUTE ALTER SCHEMA COMMAND
1839 FAILED TO EXECUTE GRANT COMMAND

Errores de Caché

2000 CACHEABLE NOT INITIALIZED
2001 VALUE ALREADY CACHED
2002 CACHED VALUE NOT FOUND
2003 CACHE IS FULL
2004 CACHING IS NOT POSSIBLE

Errores de protocolo

3000 HEADER NOT FOUND
3001 UNKNOWN COMMAND
3002 ALREADY LOGGED IN
3003 NOT LOGGED IN
3004 UNKNOWN OUTPUT MODE
3005 INVALID STATEMENT ID
3006 UNKNOWN DATA TYPE
3007 STILL LOGGED IN
3008 SOCKET READ ERROR
3009 SOCKET WRITE ERROR
3010 BASE64 DECODING ERROR
3011 SESSION TIMEOUT
3012 FETCH TIMESTAMP ALREADY EXISTS
3013 BASE64 ENCODING ERROR
3014 INVALID HEADER TERMINATOR
3015 INVALID SESSION TICKET
3016 HEADER TOO LONG
3017 INVALID AGENT SIGNATURE
3018 UNEXPECTED HEADER VALUE

Errores de red

La siguiente tabla describe los errores que pueden ocurrir en conexiones de red.

Código	***Descripción***
-10051	*Valor incorrecto para la opción de componente de red.*
-10050	*Opción de componente de red desconocido.*
-10033	*Tamaño de datos incorrecto durante el ciclo de lectura.*
-10031	*Desincronización durante el ciclo de lectura.*
-10030	*Desincronización durante el ciclo de escritura.*
-10021	*Ningún servidor encontrado.*
-10020	*Ningún servidor seleccionado.*
-10003	*Parámetros de conexión incorrectos.*
-10002	*La conexión para este proceso ha sido interrumpida o no se pudo establecer la conexión.*
-10001	*Interrupción de la conexión actual a la base.*

Errores de gestión de backup

La siguiente tabla lista los códigos de error específicos generados por el módulo de backup y de restitución de 4D. Puede interceptar estos errores utilizando un método instalado vía el comando ***ON ERR CALL****.*

Código	*Descripción*
1401	*The maximum number of backup attempts has been reached; automatic backup is temporarily disabled.*
1403	*No log file.*
1404	*A transaction is opened in this process.*
1405	*The maximum timeout for transactions to end in a concurrent process has been reached.*
1406	*Backup canceled by user.*
1407	*Destination folder is not valid.*
1408	*Error during log file backup.*
1409	*Error during backup.*
1410	*Cannot find the backup file to be checked.*
1411	*Error during backup file check.*
1412	*Cannot find the log backup file to be checked.*
1413	*Error during log backup file check.*
1414	*This comando can only be executed on 4D Server.*
1415	*Cannot back up log file; a critical operation is in progress.*
1416	*This log file does not correspond to the database opened.*
1417	*A log integration operation is already running. The backup cannot be launched.*
1420	*Integration aborted due to detection of locked records.*
1421	*This command cannot be used in a client/server environment.*

- *Los errores 1408 y 1409 provienen generalmente de un error de lectura de los archivos a los cuales se les va a realizar una copia de seguridad o a un error de escritura con los archivos a los que se les está haciendo backup.*
- *Los errores 1411 y 1413 se presentan durante la verificación de los archivos.*

Cuando se presentan estos errores, es prudente verificar primero el espacio disponible en el disco y los privilegios de acceso de lectura-escritura.

Errores de actualización cliente (-10650 -> -10655)

La siguiente tabla describen los errores que pueden ocurrir en el contexto de una actualización cliente:

Código	***Descripción***
-10650	*No puede descargar la actualización cliente*
-10651	*No puede expandir la actualización cliente*
-10652	*La actualización de la información no está disponible ({ruta})*
-10653	*No se puede descargar la actualización '{ruta}'*
-10654	*Actualización de información no válida*
-10655	*La actualización cliente no está disponible*

Errores del Actualizador (-10603 -> -10613)

La siguiente tabla describe los errores del actualizador que pueden ocurrir:

Código	***Descripción***
-10603	*La instalación del actualizador falló*
-10604	*No se puede identificar el paquete de actualización {ruta}*
-10605	*Error al copiar archivos del paquete actualizador*
-10606	*No se puede deshabilitar la versión de actualización actual*
-10607	*No se puede crear una nueva carpeta de instalación de actualización*
-10608	*No se puede encontrar la carpeta que contiene el software de actualización {ruta}*
-10609	*No se puede iniciar el actualizador de software*
-10611	*No se puede iniciar el actualizador de software (instalación no válida)*
-10612	*La aplicación en ejecución no puede actualizar con ella misma*
-10613	*No se puede crear dos ventanas de formulario de tipo barra de herramientas*

Errores de gestión de archivos del SO

La siguiente tabla lista los códigos de error devueltos por el administrador de archivos del sistema operativo. Estos códigos pueden aparecer cuando usted está utilizando, por ejemplo, los comandos del tema Documentos sistema. En esta lista, la palabra "archivo" indica un documento en disco y no un archivo en su estructura de base.

Código	***Descripción***
-33	*File directory full. You cannot create new files on disk.*
-34	*Disk is full. There is no more room available on the disk.*
-35	*Specified volume doesn't exist.*
-36	*I/O error. There is probably a bad block on the disk.*
-37	*Bad filename or volume name.*
-38	*Tried to read or write to a file that is not open.*
-39	*Logical end-of-file reached during read operation.*
-40	*Attempt to position before start of file.*
-41	*Not enough memory to open a new file on the disk.*
-42	*Too many files open at the same time.*
-43	*File not found.*
-44	*Volume is locked by a hardware setting.*
-45	*The file is locked, or the pathname is not correct.*
-46	*Volume is locked by an application.*
-47	*The file is already open, or the folder is not empty.*
-48	*Tried to rename a file with the name of an already deleted file.*
-49	*Tried to open a file already open with write permission.*
-51	*Tried to access a document with an invalid document reference number.*
-52	*Internal file manager error (position of file marker is lost).*
-53	*Volume not on line.*
-54	*Attempt to open locked file for writing.*
-57	*Tried to work with a non-Macintosh disk.*
-58	*Error in the external file system.*
-60	*Bad master directory block. Your disk is damaged.*
-61	*Read/write permission doesn't allow writing.*
-64	*There is a hardware problem with the disk (bad installation, incorrect formatting,...).*
-84	*There is a hardware problem with the disk (bad installation, incorrect formatting,...).*
-120	*Tried to access a file by using a pathname that specifies a non existing directory.*
-121	*An access path could not be created.*
-124	*Tried to access a disconnected shared volume.*

Errores de gestión de memoria del SO

La siguiente tabla lista los principales códigos de error devueltos por el administrador de memoria del sistema operativo.

Código	***Descripción***
-108	*Not enough memory to perform an operation.* *Give more memory to your 4D application.*
-109	*Internal Memory problem. Memory is probably logically corrupted.* *Exit as soon as possible. Restart your machine and reopen the database.*
-111	*Internal Memory problem. Memory is probably logically corrupted.* *Exit as soon as possible. Restart your machine and reopen the database. (*)*
-117	*Internal Memory problem. Memory is probably logically corrupted.* *Exit as soon as possible. Restart your machine and reopen the database.*

Consejo: *cuando trabaja con arrays, BLOBs, imágenes o conjuntos muy grandes (es decir objetos que puedan manipular grandes cantidades de datos), utilice un método de proyecto instalado por* ***ON ERR CALL*** *para probar el error -108.*

(*) Un error -111 también se puede producir cuando intenta leer un valor en un BLOB con un offset fuera del rango. En este caso, el error es menor y no necesita terminar la sesión de trabajo. Tan solo corrija el offset que pasa al comando BLOB.

Errores de gestión de impresión del SO

La siguiente tabla lista los códigos de error devueltos por el administrador de impresión del sistema operativo. Estos códigos pueden ser devueltos durante la impresión.

Código	***Descripción***
-1	*Problem saving file to be printed*
-27	*Problem opening or closing connection with printer*
-128	*Printing interrupted by the user*
-193	*Resource file not found*
-4100	*Printer connection has been interrupted*
-4101	*Printer busy or not connected*
-8150	*A LaserWriter is not selected*
-8151	*The printer has been initialized with a different driver version*
-8192	*LaserWriter time-out*

Errores de gestión de recursos del sistema

La siguiente tabla lista los códigos devueltos por el administrador de recursos del sistema operativo.

Código	***Descripción***
-1	*Resource file could not be opened*
-192	*Resource not found*
-193	*Resource map is damaged (file needs to be repaired)*
-194	*Resource could not be added*
-196	*Resource could not be deleted*

Errores de gestión del sonido del sistema

La siguiente tabla lista los códigos devueltos por el administrador del sonido del sistema operativo

Código	***Descripción***
-203	*Demasiados comandos de sonido*
-204	*No se puede cargar el recurso de sonido*
-205	*El canal de sonido está dañado*
-206	*El formato del recurso de sonido es incorrecto*
-207	*Memoria insuficiente para reproducir el sonido*
-209	*El canal de sonido no está disponible*

Errores de gestión del puerto serial del sistema

La siguiente tabla lista el código de error devuelto por el administrador de puerto serial del sistema operativo.

Código	***Descripción***
-28	*No hay puerto serial abierto*

Errores Mac OS

La siguiente tabla lista algunos errores devueltos por el sistema Mac OS. Generalmente no es posible recuperarse de estos errores.

Código	***Descripción***
4	*Zero divide*
15	*Segment Loader Error: 4D failed in loading one of its own code segments.* *You must allocate more memory to 4D.*
17 a 24	*A system package is missing. Check if your system directory has been correctly installed*
25	*Out of memory* *You must allocate more memory to 4D.*
28	*Stack has moved into the application heap.* *You must allocate more memory to 4D.*

Errores varios (-10700)

La siguiente tabla describe los errores varios que pueden ocurrir:

Código	***Descripción***
-10700	*El puerto debe estar en el rango de 0 a 65535.*

Códigos ASCII

Códigos Unicode
Códigos ASCII
Códigos de teclas de función

Códigos Unicode

En 4D, el lenguaje así como el motor de la base de datos almacenan y manipulan nativamente los caracteres en Unicode. Este principio facilita la internacionalización de las aplicaciones 4D. El unicode es un juego de caracteres estándar unificado que administra prácticamente todo los lenguajes utilizados en el mundo. Un juego de caracteres es una tabla de correspondencia caracter/valor numérico, por ejemplo "a"->1, "b"->2, "5"->15, "oe"->662, etc. Mientras que con ASCII, el valor numérico de base está generalmente entre 1 y 127, con Unicode el límite superior supera 65 000, lo que significa que puede representar casi todos los caracteres de todos los lenguajes.

Hay diferentes maneras de codificar los valores numéricos Unicode: UTF-16 los codifica en enteros de 16-bits, UTF-32 utiliza enteros de 32-bits y UTF-8 enteros de 8-bits. 4D utiliza principalmente UTF-16 (como Windows y Mac OS). Algunas veces, esencialmente para necesidades específicas relacionadas con Internet, 4D utiliza UTF-8 que tiene la ventaja de ser más compacto y tener mejor legibilidad para los caracteres comunes (a-z,0-9).

Nota: *4D usa internamente la codificación UTF-16 para almacenar texto en campos y variables (incluso si la base se está ejecutando en modo no Unicode). A menos que se mencione específicamente, un carácter, posición o longitud manejada a través del lenguaje de programación siempre se refiere a los valores en UTF-16.*

Para mayor información sobre el Unicode estándar, por favor consulte, la siguiente página:
http://en.wikipedia.org/wiki/Unicode

Una lista de códigos Unicode:
http://en.wikipedia.org/wiki/List_of_Unicode_characters#See_also

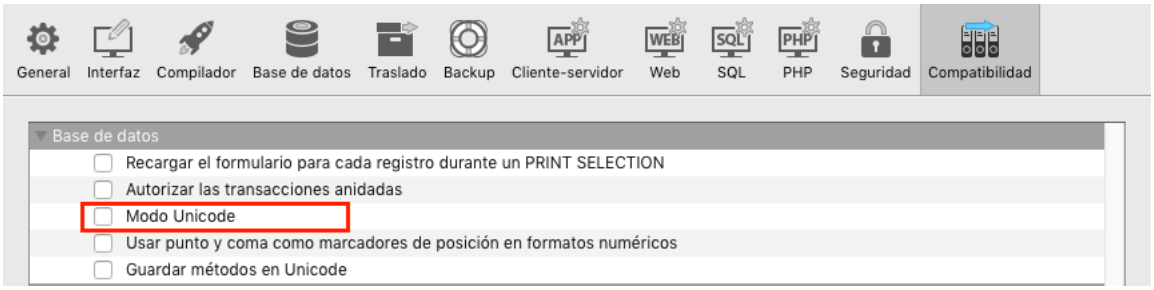
Advertencia: *en Unicode en 4D v11, los siguientes códigos de caracteres están reservados y no deben incluirse en un texto:*
0
65534 (FFFE)
65535 (FFFF)

Nota de compatibilidad: *las bases creadas con una versión 4D anterior a la versión 11 pueden funcionar en modo compatibilidad ASCII. Para mayor información, por favor consulte la sección* ***EXPORT TEXT****.*

Códigos ASCII

Nota de compatibilidad: *4D puede funcionar con dos conjuntos de caracteres: Unicode o ASCII. El modo Unicode es el modo estándar a partir de la versión 11 de 4D. El modo ASCII se conserva por motivos de compatibilidad en bases creadas con versiones anteriores de 4D. Este modo se llama* ***modo compatibilidad ASCII****.*

Es posible activar el modo Unicode en las bases de datos convertidas vía el selector Unicode Mode de los comandos ***Get database parameter*** *y* ***SET DATABASE PARAMETER*** *o vía la opción* ***Modo Unicode*** *que se encuentra en la página Compatibilidad de las Propiedades de la base:*



En la mayoría de los casos, el funcionamiento inicial de las aplicaciones no se ve afectado por esta configuración, ya que 4D maneja cualquier conversión de caracteres internamente. Además, los caracteres más comunes (a-z, 0-9, etc.) tienen el mismo valor (de 1 a 127) en ambos Unicode y ASCII (Windows y Mac OS).
Sin embargo, algunas instrucciones de lenguaje, particularmente las que utilizan comandos que funcionan con cadenas de caracteres, podrían requerir alguna adaptación. Por ejemplo, la instrucción ***Char****(200) no devolverá el mismo valor en Unicode que en ASCII. Este manual describe las diferencias de funcionamiento entre el modo Unicode y el modo de compatibilidad ASCII para cada comando.*

Nota: *este modo se especifica para cada base de datos. Por lo tanto es posible tener una base Unicode coexistiendo con los componentes no-Unicode (o viceversa).*

4D y los códigos ASCII

Cuando la base funciona en modo compatibilidad ASCII, en Mac OS y Windows, el motor interno de la base y el lenguaje de 4D trabajan con el mapa ASCII extendido de Macintosh. Cuando introduce datos utilizando el teclado (adición de registros, edición de métodos, etc.), 4D utiliza el esquema interno de conversión de Altura ASCII para convertir los códigos provenientes del teclado (que son expresados con la ayuda del mapa ASCII extendido Windows) en códigos Macintosh. Por ejemplo, para introducir una "é", digite ALT+0233 y 4D almacena el código ASCII 142 en el registro. Esto es transparente para el usuario final, por ejemplo cuando usted efectúa una búsqueda, introduce el valor real a buscar en el editor de búsquedas. Por lo tanto, el valor que digitó (ALT+0233) también se traduce en código ASCII 142 y la búsqueda termina.

El mismo principio se aplica cuando digita ALT+0233 en el editor de métodos. Sin embargo, para buscar un carácter utilizando su código ASCII, debe utilizar el código ASCII Macintosh del carácter.

Por ejemplo:

```
QUERY(...;[MiArchivo]MiCampo="é") ` é es Alt+0233
```

es idéntico a:

```
QUERY(...;[MiArchivo]MiCampo=Char(142)) ` é es ASCII 142
```

Tablas de códigos ASCII

- *La tabla estándar de los códigos ASCII (0 a 127), es idéntica para las plataformas Windows y Macintosh. Estos códigos ASCII estándar se listan en ASCII Codes 0..63 y ASCII Codes 64..127.*
- *La tabla ASCII (códigos ASCII de 128 a 255) es diferente en Windows y Macintosh. Con el fin de asegurar la independencia de plataforma de sus aplicaciones, 4D convierte automáticamente los códigos ASCII (del mapa Windows al mapa Mac OS) cuando los caracteres se introducen en el entorno 4D (entrada de datos, copiar/pegar, importar registros, etc.) y (del mapa Mac OS al mapa Window) cuando los caracteres se extraen del entorno 4D (cortar/pegar, exportar, etc.).*
  Los códigos ASCII del 128 al 255 se listan en las secciones ASCII Codes 128..191 y ASCII Codes 192..255.
- *Códigos ASCII 0 a 63*

ASCII		Macintosh or Windows
Dec	Hex	Result
0	0	*NUL*
1	1	*SOH*
2	2	*STX*
3	3	*ETX*
4	4	*EOT*
5	5	*ENQ*
6	6	*ACK*
7	7	*BEL*
8	8	*BS*
9	9	*HT*
10	A	*LF*
11	B	*VT*
12	C	*FF*
13	D	*CR*
14	E	*SO*
15	F	*SI*

ASCII		Macintosh or Windows
Dec	Hex	Result
16	10	*DLE*
17	11	*DC1*
18	12	*DC2*
19	13	*DC3*
20	14	*DC4*
21	15	*NAK*
22	16	*SYN*
23	17	*ETB*
24	18	*CAN*
25	19	*EM*
26	1A	*SUB*
27	1B	*ESC*
28	1C	*FS*
29	1D	*GS*
30	1E	*RS*
31	1F	*US*

ASCII		Macintosh or Windows
Dec	Hex	Result
32	20	*sp*
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/

ASCII		Macintosh or Windows
Dec	Hex	Result
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

• *Códigos ASCII 64 a 127*

ASCII		Macintosh or Windows
Dec	Hex	Result
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O

ASCII		Macintosh or Windows
Dec	Hex	Result
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

ASCII		Macintosh or Windows
Dec	Hex	Result
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

ASCII		Macintosh or Windows
Dec	Hex	Result
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

- *Códigos ASCII 128 a 191*

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
128	80	Ä	Option-u, Shift-a	Ä	Alt+0196
129	81	Å	Option-Shift-a	Å	Alt+0197
130	82	Ç	Option-Shift-c	Ç	Alt+0199
131	83	É	Option-e, Shift-e	É	Alt+0201
132	84	Ñ	Option-n, Shift-n	Ñ	Alt+0209
133	85	Ö	Option-u, Shift-o	Ö	Alt+0214
134	86	Ü	Option-u, Shift-u	Ü	Alt+0220
135	87	á	Option-e, a	á	Alt+0225
136	88	à	Option-`, a	à	Alt+0224
137	89	â	Option-i, a	â	Alt+0226
138	8A	ä	Option-u, a	ä	Alt+0228
139	8B	ã	Option-n, a	ã	Alt+0227
140	8C	å	Option-a	å	Alt+0229
141	8D	ç	Option-c	ç	Alt+0231
142	8E	é	Option-e, e	é	Alt+0233
143	8F	è	Option-`, e	è	Alt+0232

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
144	90	ê	Option-i, e	ê	Alt+0234
145	91	ë	Option-u, e	ë	Alt+0235
146	92	í	Option-e, i	í	Alt+0237
147	93	ì	Option-`, i	ì	Alt+0236
148	94	î	Option-i, i	î	Alt+0238
149	95	ï	Option-u, i	ï	Alt+0239
150	96	ñ	Option-n, n	ñ	Alt+0241
151	97	ó	Option-e, o	ó	Alt+0243
152	98	ò	Option-`, o	ò	Alt+0242
153	99	ô	Option-i, o	ô	Alt+0244
154	9A	ö	Option-u, o	ö	Alt+0246
155	9B	õ	Option-n, o	õ	Alt+0245
156	9C	ú	Option-e, u	ú	Alt+0250
157	9D	ù	Option-`, u	ù	Alt+0249
158	9E	û	Option-i, u	û	Alt+0251
159	9F	ü	Option-u, u	ü	Alt+0252

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
160	A0	†	Shift-Option-7		
161	A1	°	Shift-Option-7	°	Alt+0176
162	A2	¢	Option-4	¢	Alt+0162
163	A3	£	Option-3	£	Alt+0163
164	A4	§	Option-6	§	Alt+0167
165	A5	•	Option-8		
166	A6	¶	Option-7	¶	Alt+0182
167	A7	ß	Option-s	ß	Alt+0223
168	A8	®	Option-r	®	Alt+0174
169	A9	©	Option-g	©	Alt+0169
170	AA	™	Option-2	™	Alt-0153
171	AB	´	Shift-Option-e	´	Alt+0145
172	AC	¨	Shift-Option-u	¨	Alt+0168
173	AD	≠	Option-=		
174	AE	Æ	Shift-Option-"	Æ	Alt+0198
175	AF	Ø	Shift-Option-o	Ø	Alt+0216

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
176	B0	∞	Option-5		
177	B1	±	Shift-Option-=	±	Alt+0177
178	B2	≤	Option-,		
179	B3	≥	Option-. (period)		
180	B4	¥	Option-y	¥	Alt+0165
181	B5	µ	Option-m	µ	Alt+0181
182	B6	∂	Option-d		
183	B7	∑	Option-w		
184	B8	∏	Shift-Option-p		
185	B9	π	Option-p		
186	BA	∫	Option-b		
187	BB	ª	Option-9	ª	Alt+0170
188	BC	º	Option-0 (zero)	º	Alt+0186
189	BD	Ω	Option-z		
190	BE	æ	Option-'	æ	Alt+0230
191	BF	ø	Option-o	ø	Alt+0248

- *Códigos ASCII 192 a 255*

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
192	C0	¿	Shift-Option-?	¿	Alt+0191
193	C1	¡	Option-1	¡	Alt+0161
194	C2	¬	Option-l (L)	¬	Alt+0172
195	C3	√	Option-v		
196	C4	ƒ	Option-f		
197	C5	≈	Option-x		
198	C6	Δ	Option-j		
199	C7	«	Option-\	«	Alt+0171
200	C8	»	Shift-Option-\	»	Alt+0187
201	C9	…	Option-;	…	Alt+0133
202	CA	(space)	Spacebar	(space)	Alt+0160
203	CB	À	Option-`, Shift-a	À	Alt+0192
204	CC	Ã	Option-n, Shift-a	Ã	Alt+0195
205	CD	Õ	Option-n, Shift-o	Õ	Alt+0213
206	CE	Œ	Shift-Option-q		
207	CF	œ	Option-q		

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
208	D0	–	Option--(dash)		
209	D1	—	Shift-Option-(dash)		
210	D2	“	Option-[	“	Alt+0147
211	D3	”	Shift-Option-[	”	Alt+0148
212	D4	‘	Option-]	‘	Alt+0145
213	D5	’	Shift-Option-]	’	Alt+0146
214	D6	÷	Option-/	÷	Alt+0247
215	D7	◊	Shift-Option-v		
216	D8	ÿ	Option-u, y	ÿ	Alt+0255
217	D9	Ÿ	Option-u, Shift-y		
218	DA	⁄	Shift-Option-1		
219	DB	€	Shift-Option-2	¤	Alt+0164
220	DC	‹	Shift-Option-3		
221	DD	›	Shift-Option-4		
222	DE	fi	Shift-Option-5		
223	DF	fl	Shift-Option-6		

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
224	E0	‡	Shift-Option-7		
225	E1	·	Shift-Option-9	·	Alt+0183
226	E2	‚	Shift-Option-0		
227	E3	„	Shift-Option-w		
228	E4	‰	Shift-Option-r		
229	E5	Â	Option-i, Shift-a	Å	Alt+0194
230	E6	Ê	Option-i, Shift-e	Ê	Alt+0202
231	E7	Á	Option-e, Shift-a	Á	Alt+0193
232	E8	Ë	Option-u, Shift-e	Ë	Alt+0203
233	E9	È	Option-`, Shift-e	È	Alt+0200
234	EA	Í	Shift-Option-s	Í	Alt+0205
235	EB	Î	Shift-Option-d	Î	Alt+0206
236	EC	Ï	Shift-Option-f	Ï	Alt+0207
237	ED	Ì	Option-`, Shift-i	Ì	Alt+0204
238	EE	Ó	Shift-Option-h	Ó	Alt+0211
239	EF	Ô	Shift-Option-j	Ô	Alt+0212

ASCII		Macintosh		Windows	
Dec	Hex	Result (in Times)	Key Combination	Result (in Arial)	Key Combination
240	F0		Shift-Option-k		
241	F1	Ò	Shift-Option-l (L)	Ò	Alt+0210
242	F2	Ú	Shift-Option-;	Ú	Alt+0218
243	F3	Û	Option-i, Shift-u	Û	Alt+0219
244	F4	Ù	Option-`, Shift-u	Ù	Alt+0217
245	F5	ı	Shift-Option-b		
246	F6	ˆ	Shift-Option-i		
247	F7	˜	Shift-Option-n		
248	F8	¯	Shift-Option-,	¯	Alt+0175
249	F9	˘	Shift-Option-.		
250	FA	˙	Option-h		
251	FB	˚	Option-k		
252	FC	¸	Shift-Option-z	¸	Alt+0184
253	FD	˝	Shift-Option-g		
254	FE	˛	Shift-Option-x		
255	FF	ˇ	Shift-Option-t		

Nota: *las cajas grises indican los caracteres que no están disponibles bajo Window o que difieren de los caracteres Macintosh.*

Códigos de teclas de función

---

*4D devuelve valores para teclas de función en la variable sistema **KeyCode**, que se utiliza en los métodos de proyecto instalados por el comando **ON EVENT CALL**. Estos métodos de proyecto se utilizan para interceptar eventos.*

Los valores de las teclas de función no están basados en códigos ASCII y son:

Function key	KeyCode
F1	-122
F2	-120
F3	-99
F4	-118
F5	-96
F6	-97
F7	-98
F8	-100
F9	-101
F10	-109
F11	-103
F12	-111
F13	-105
F14	-107
F15	-113

Recuerde

*La variable sistema **KeyCode** se utiliza en un método de proyecto instalado utilizando **ON EVENT CALL**.*

*Además de las teclas de función, la siguiente tabla lista los valores devueltos en **KeyCode** cuando presiona una de las teclas estándar como Retorno de carro o Enter.*

Key	Code
Enter	3
Return	13
Backspace	8
Tab	9
Escape	27
Del	127
Help	5
Home	1
End	4
Page Up	11
Page Down	12
Left Arrow	28
Right Arrow	29
Up Arrow	30
Down Arrow	31

Manual de lenguaje 4D - Comandos obsoletos

17.0

- _o_ADD DATA SEGMENT
- _o_ADD SUBRECORD
- _o_ALL SUBRECORDS
- _o_APPLY TO SUBSELECTION
- _o_ARRAY STRING
- _o_ARRAY TO STRING LIST
- _o_Before subselection
- _o_C_GRAPH
- _o_C_INTEGER
- _o_C_STRING
- _o_Convert case
- _o_Create resource file
- _o_CREATE SUBRECORD
- _o_DATA SEGMENT LIST
- _o_DELETE RESOURCE
- _o_DELETE SUBRECORD
- _o_DISABLE BUTTON
- _o_Document creator
- _o_Document type

- *_o_During*
- *_o_ENABLE BUTTON*
- *_o_End subselection*
- *_o_FIRST SUBRECORD*
- *_o_Font name*
- *_o_Font number*
- *_o_Gestalt*
- *_o_Get component resource ID*
- *_o_Get platform interface*
- *_o_GRAPH TABLE*
- *_o_INTEGRATE LOG FILE*
- *_o_INVERT BACKGROUND*
- *_o_ISO to Mac*
- *_o_LAST SUBRECORD*
- *_o_Mac to ISO*
- *_o_Mac to Win*
- *_o_MAP FILE TYPES*
- *_o_MODIFY SUBRECORD*
- *_o_NEXT SUBRECORD*
- *_o_OBJECT Get action*
- *_o_Open external window*
- *_o_ORDER SUBRECORDS BY*
- *_o_PICTURE TO GIF*
- *_o_PICTURE TYPE LIST*
- *_o_PLATFORM PROPERTIES*
- *_o_PREVIOUS SUBRECORD*
- *_o_QT COMPRESS PICTURE*
- *_o_QT COMPRESS PICTURE FILE*
- *_o_QT LOAD COMPRESS PICTURE FROM FILE*
- *_o_QUERY SUBRECORDS*
- *_o_Records in subselection*
- *_o_REDRAW LIST*
- *_o_SAVE PICTURE TO FILE*
- *_o_SET CGI EXECUTABLE*
- *_o_SET DOCUMENT CREATOR*
- *_o_SET DOCUMENT TYPE*
- *_o_SET PICTURE RESOURCE*
- *_o_SET PLATFORM INTERFACE*
- *_o_SET RESOURCE*
- *_o_SET RESOURCE NAME*
- *_o_SET RESOURCE PROPERTIES*
- *_o_SET STRING RESOURCE*
- *_o_SET TEXT RESOURCE*
- *_o_SET WEB DISPLAY LIMITS*
- *_o_SET WEB TIMEOUT*
- *_o_USE EXTERNAL DATABASE*
- *_o_USE INTERNAL DATABASE*
- *_o_Web Context*
- *_o_Win to Mac*
- *_o_XSLT APPLY TRANSFORMATION*
- *_o_XSLT GET ERROR*
- *_o_XSLT SET PARAMETER*